FLCL - User Manual

COLLABORATORS

	TITLE : FLCL - User Manual	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	limes datentechnik gmbh	Aug 21 2015	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
5.1.8	Aug 21 2015	released	LDG

Contents

1 Abstract **1**

2 COMMAND LINE PROCESSOR **4**

2.1 USED ENVIRONMENT VARIABLES 4

2.2 FILENAME MAPPING 5

2.3 SPECIAL EBCDIC CODE PAGE SUPPORT 6

3 PROGRAM *flcl* **7**

3.1 SYNOPSIS 7

3.2 DESCRIPTION 7

3.2.1 USED ENVIRONMENT VARIABLES 8

3.2.2 FILENAME HANDLING 9

3.2.3 DIRECTORY SUPPORT 10

3.2.4 INPUT TO OUTPUT NAME MAPPING 11

3.2.5 USE OF BUILT-IN FUNCTIONS 13

3.2.6 CONV versus XCNV and ICNV 14

3.3 SYNTAX 15

3.4 HELP 16

4 Available commands **18**

4.1 COMMAND *INFO* 18

4.1.1 OVERLAY *GET* 19

4.1.2 OBJECT *FKME* 20

4.1.3 PARAMETER *OUT* 20

4.1.4 OBJECT *DIR* 21

4.1.5 OBJECT *LOG* 21

4.1.6 OBJECT *MESSAGE* 22

4.1.7 OVERLAY *DESTINATION* 23

4.1.8 OBJECT *STREAM* 23

4.1.9 OBJECT *FILE* 23

4.1.10 OBJECT *SYSTEM* 24

4.2 COMMAND *CONV* 24
4.2.1 OVERLAY *READ* 26
4.2.2 OBJECT *BINARY* 26
4.2.3 PARAMETER *BLKSIZE* 27
4.2.4 OVERLAY *DECRYPT* 28
4.2.5 OBJECT *F4PWD* 28
4.2.6 OBJECT *F4KME* 29
4.2.7 OVERLAY *FDECODE* 30
4.2.8 PARAMETER *PRNCONTROL* 30
4.2.9 OBJECT *SUBSYSTEM* 31
4.2.10 PARAMETER *PLATFORM* 31
4.2.11 OBJECT *HASH* 32
4.2.12 OVERLAY *CHECK* 33
4.2.13 PARAMETER *CHECK* 34
4.2.14 OBJECT *CHARACTER* 34
4.2.15 PARAMETER *BLKSIZE* 35
4.2.16 PARAMETER *CCSID* 35
4.2.17 PARAMETER *BOM* 37
4.2.18 PARAMETER *ENL2LF* 37
4.2.19 OVERLAY *DECRYPT* 37
4.2.20 OBJECT *F4PWD* 38
4.2.21 OBJECT *F4KME* 38
4.2.22 PARAMETER *PRNCONTROL* 39
4.2.23 OBJECT *SUBSYSTEM* 39
4.2.24 PARAMETER *LANG* 40
4.2.25 PARAMETER *PLATFORM* 40
4.2.26 OBJECT *HASH* 41
4.2.27 OVERLAY *CHECK* 42
4.2.28 PARAMETER *CHECK* 43
4.2.29 OBJECT *TEXT* 43
4.2.30 PARAMETER *BLKSIZE* 45
4.2.31 PARAMETER *CCSID* 45
4.2.32 PARAMETER *BOM* 46
4.2.33 PARAMETER *ENL2LF* 47
4.2.34 OVERLAY *DECRYPT* 47
4.2.35 OBJECT *F4PWD* 47
4.2.36 OBJECT *F4KME* 48
4.2.37 PARAMETER *PRNCONTROL* 49
4.2.38 OBJECT *SUBSYSTEM* 49

4.2.39 PARAMETER *LANG* . . . 50
4.2.40 PARAMETER *PLATFORM* . . . 50
4.2.41 OBJECT *HASH* . . . 51
4.2.42 OVERLAY *CHECK* . . . 52
4.2.43 PARAMETER *CHECK* . . . 52
4.2.44 OBJECT *XML* . . . 53
4.2.45 PARAMETER *BLKSIZE* . . . 54
4.2.46 PARAMETER *CCSID* . . . 54
4.2.47 PARAMETER *BOM* . . . 55
4.2.48 PARAMETER *ENL2LF* . . . 56
4.2.49 OVERLAY *DECRYPT* . . . 56
4.2.50 OBJECT *F4PWD* . . . 56
4.2.51 OBJECT *F4KME* . . . 57
4.2.52 OBJECT *SUBSYSTEM* . . . 58
4.2.53 PARAMETER *LANG* . . . 58
4.2.54 PARAMETER *PLATFORM* . . . 59
4.2.55 OBJECT *HASH* . . . 59
4.2.56 OVERLAY *CHECK* . . . 60
4.2.57 PARAMETER *CHECK* . . . 61
4.2.58 OBJECT *RECORD* . . . 61
4.2.59 PARAMETER *PRNCONTROL* . . . 65
4.2.60 PARAMETER *BLKSIZE* . . . 66
4.2.61 OVERLAY *LENFORMAT* . . . 66
4.2.62 OBJECT *INTEGER* . . . 67
4.2.63 OBJECT *STRING* . . . 67
4.2.64 OBJECT *BCD* . . . 68
4.2.65 OBJECT *HOST* . . . 68
4.2.66 OVERLAY *DECRYPT* . . . 69
4.2.67 OBJECT *F4PWD* . . . 69
4.2.68 OBJECT *F4KME* . . . 70
4.2.69 OVERLAY *FDECODE* . . . 70
4.2.70 PARAMETER *CCSID* . . . 71
4.2.71 PARAMETER *BOM* . . . 72
4.2.72 OBJECT *SUBSYSTEM* . . . 72
4.2.73 PARAMETER *LANG* . . . 73
4.2.74 PARAMETER *PLATFORM* . . . 73
4.2.75 OBJECT *HASH* . . . 74
4.2.76 OVERLAY *CHECK* . . . 75
4.2.77 PARAMETER *CHECK* . . . 76

4.2.78 OBJECT *FLAM4* . . . 76
4.2.79 PARAMETER *PRNCONTROL* . . . 77
4.2.80 PARAMETER *CCSID* . . . 77
4.2.81 PARAMETER *BOM* . . . 79
4.2.82 PARAMETER *LANG* . . . 79
4.2.83 PARAMETER *PLATFORM* . . . 79
4.2.84 OBJECT *HASH* . . . 80
4.2.85 OVERLAY *CHECK* . . . 81
4.2.86 OBJECT *AUTO* . . . 82
4.2.87 PARAMETER *BLKSIZE* . . . 83
4.2.88 PARAMETER *CCSID* . . . 83
4.2.89 PARAMETER *BOM* . . . 84
4.2.90 PARAMETER *ENL2LF* . . . 85
4.2.91 OVERLAY *DECRYPT* . . . 85
4.2.92 OBJECT *F4PWD* . . . 85
4.2.93 OBJECT *F4KME* . . . 86
4.2.94 PARAMETER *LANG* . . . 87
4.2.95 PARAMETER *PLATFORM* . . . 87
4.2.96 OBJECT *HASH* . . . 88
4.2.97 OVERLAY *CHECK* . . . 89
4.2.98 PARAMETER *CHECK* . . . 89
4.2.99 PARAMETER *FILE* . . . 90
4.2.100 OVERLAY *WRITE* . . . 90
4.2.101 OBJECT *BINARY* . . . 90
4.2.102 OBJECT *FALLOC* . . . 91
4.2.103 OBJECT *SUBSYSTEM* . . . 94
4.2.104 OBJECT *SPACE* . . . 94
4.2.105 OVERLAY *COMPRESS* . . . 95
4.2.106 OBJECT *GZIP* . . . 96
4.2.107 OBJECT *BZIP2* . . . 96
4.2.108 OBJECT *XZ* . . . 97
4.2.109 OBJECT *FLAM4* . . . 98
4.2.110 OVERLAY *ENCRYPT* . . . 98
4.2.111 OBJECT *F4PWD* . . . 99
4.2.112 OBJECT *F4KME* . . . 99
4.2.113 OVERLAY *ENCODE* . . . 100
4.2.114 OBJECT *BASE64* . . . 101
4.2.115 OBJECT *ARMOR* . . . 102
4.2.116 OBJECT *BASE32* . . . 104

4.2.117 OBJECT *ARMOR* . . . 106
4.2.118 OBJECT *BASE16* . . . 107
4.2.119 OBJECT *ARMOR* . . . 109
4.2.120 OBJECT *OPGP* . . . 110
4.2.121 PARAMETER *PRNCONTROL* . . . 112
4.2.122 PARAMETER *PLATFORM* . . . 112
4.2.123 OBJECT *HASH* . . . 113
4.2.124 OVERLAY *CHECK* . . . 114
4.2.125 OBJECT *CHARACTER* . . . 115
4.2.126 OBJECT *FALLOC* . . . 116
4.2.127 OBJECT *SUBSYSTEM* . . . 119
4.2.128 OBJECT *SPACE* . . . 119
4.2.129 PARAMETER *CCSID* . . . 120
4.2.130 PARAMETER *CASE* . . . 121
4.2.131 PARAMETER *BOM* . . . 122
4.2.132 PARAMETER *CHRMODE* . . . 122
4.2.132.1 CONSTANT *STOP* . . . 122
4.2.132.2 CONSTANT *IGNORE* . . . 123
4.2.132.3 CONSTANT *SUBSTITUTE* . . . 123
4.2.133 PARAMETER *ELF2NL* . . . 123
4.2.134 PARAMETER *SUBCHAR* . . . 123
4.2.135 PARAMETER *SYSTABLE* . . . 124
4.2.136 PARAMETER *USRTABLE* . . . 124
4.2.137 PARAMETER *REPORTFILE* . . . 127
4.2.138 OVERLAY *COMPRESS* . . . 128
4.2.139 OBJECT *GZIP* . . . 129
4.2.140 OBJECT *BZIP2* . . . 129
4.2.141 OBJECT *XZ* . . . 130
4.2.142 OBJECT *FLAM4* . . . 131
4.2.143 OVERLAY *ENCRYPT* . . . 131
4.2.144 OBJECT *F4PWD* . . . 132
4.2.145 OBJECT *F4KME* . . . 132
4.2.146 OVERLAY *ENCODE* . . . 133
4.2.147 OBJECT *BASE64* . . . 134
4.2.148 OBJECT *ARMOR* . . . 135
4.2.149 OBJECT *BASE32* . . . 137
4.2.150 OBJECT *ARMOR* . . . 139
4.2.151 OBJECT *BASE16* . . . 140
4.2.152 OBJECT *ARMOR* . . . 142

4.2.153 OBJECT *OPGP* . . . 143
4.2.154 PARAMETER *PRNCONTROL* . . . 145
4.2.155 PARAMETER *LANG* . . . 145
4.2.156 PARAMETER *PLATFORM* . . . 146
4.2.157 OBJECT *HASH* . . . 146
4.2.158 OVERLAY *CHECK* . . . 148
4.2.159 OBJECT *TEXT* . . . 148
4.2.160 OBJECT *FALLOC* . . . 150
4.2.161 OBJECT *SUBSYSTEM* . . . 153
4.2.162 OBJECT *SPACE* . . . 153
4.2.163 PARAMETER *CCSID* . . . 154
4.2.164 PARAMETER *CASE* . . . 155
4.2.165 PARAMETER *BOM* . . . 156
4.2.166 PARAMETER *ELF2NL* . . . 156
4.2.167 PARAMETER *CHRMODE* . . . 156
4.2.167.1 CONSTANT *STOP* . . . 157
4.2.167.2 CONSTANT *IGNORE* . . . 157
4.2.167.3 CONSTANT *SUBSTITUTE* . . . 157
4.2.168 PARAMETER *SUBCHAR* . . . 158
4.2.169 PARAMETER *SYSTABLE* . . . 158
4.2.170 PARAMETER *USRTABLE* . . . 158
4.2.171 PARAMETER *REPORTFILE* . . . 161
4.2.172 OVERLAY *COMPRESS* . . . 162
4.2.173 OBJECT *GZIP* . . . 163
4.2.174 OBJECT *BZIP2* . . . 163
4.2.175 OBJECT *XZ* . . . 164
4.2.176 OBJECT *FLAM4* . . . 165
4.2.177 OVERLAY *ENCRYPT* . . . 165
4.2.178 OBJECT *F4PWD* . . . 166
4.2.179 OBJECT *F4KME* . . . 166
4.2.180 OVERLAY *ENCODE* . . . 167
4.2.181 OBJECT *BASE64* . . . 168
4.2.182 OBJECT *ARMOR* . . . 169
4.2.183 OBJECT *BASE32* . . . 171
4.2.184 OBJECT *ARMOR* . . . 173
4.2.185 OBJECT *BASE16* . . . 174
4.2.186 OBJECT *ARMOR* . . . 176
4.2.187 OBJECT *OPGP* . . . 177
4.2.188 PARAMETER *PRNCONTROL* . . . 179

4.2.189 PARAMETER *LANG* . . . 179
4.2.190 PARAMETER *PLATFORM* . . . 180
4.2.191 OBJECT *HASH* . . . 180
4.2.192 OVERLAY *CHECK* . . . 182
4.2.193 OBJECT *XML* . . . 182
4.2.194 OBJECT *FALLOC* . . . 183
4.2.195 OBJECT *SUBSYSTEM* . . . 186
4.2.196 OBJECT *SPACE* . . . 186
4.2.197 OVERLAY *METHOD* . . . 188
4.2.198 OBJECT *STANDARD* . . . 188
4.2.199 OBJECT *PRETTYPRINT* . . . 188
4.2.200 OBJECT *MINIMIZED* . . . 189
4.2.201 PARAMETER *CCSID* . . . 190
4.2.202 PARAMETER *BOM* . . . 191
4.2.203 PARAMETER *ELF2NL* . . . 191
4.2.204 PARAMETER *CHRMODE* . . . 192
4.2.204.1 CONSTANT *STOP* . . . 192
4.2.204.2 CONSTANT *IGNORE* . . . 192
4.2.204.3 CONSTANT *SUBSTITUTE* . . . 193
4.2.205 PARAMETER *SUBCHAR* . . . 193
4.2.206 PARAMETER *SYSTABLE* . . . 193
4.2.207 PARAMETER *USRTABLE* . . . 194
4.2.208 PARAMETER *REPORTFILE* . . . 196
4.2.209 OVERLAY *COMPRESS* . . . 198
4.2.210 OBJECT *GZIP* . . . 198
4.2.211 OBJECT *BZIP2* . . . 199
4.2.212 OBJECT *XZ* . . . 200
4.2.213 OBJECT *FLAM4* . . . 200
4.2.214 OVERLAY *ENCRYPT* . . . 201
4.2.215 OBJECT *F4PWD* . . . 201
4.2.216 OBJECT *F4KME* . . . 202
4.2.217 OVERLAY *ENCODE* . . . 203
4.2.218 OBJECT *BASE64* . . . 203
4.2.219 OBJECT *ARMOR* . . . 205
4.2.220 OBJECT *BASE32* . . . 206
4.2.221 OBJECT *ARMOR* . . . 208
4.2.222 OBJECT *BASE16* . . . 210
4.2.223 OBJECT *ARMOR* . . . 211
4.2.224 OBJECT *OPGP* . . . 213

4.2.225 PARAMETER *LANG* . . . 215
4.2.226 PARAMETER *PLATFORM* . . . 215
4.2.227 OBJECT *HASH* . . . 216
4.2.228 OVERLAY *CHECK* . . . 217
4.2.229 OBJECT *RECORD* . . . 217
4.2.230 OBJECT *FALLOC* . . . 220
4.2.231 OBJECT *SUBSYSTEM* . . . 222
4.2.232 OBJECT *SPACE* . . . 223
4.2.233 OVERLAY *LENFORMAT* . . . 224
4.2.234 OBJECT *INTEGER* . . . 225
4.2.235 OBJECT *STRING* . . . 225
4.2.236 OBJECT *BCD* . . . 226
4.2.237 OBJECT *HOST* . . . 226
4.2.238 PARAMETER *PRNCONTROL* . . . 226
4.2.239 PARAMETER *CCSID* . . . 227
4.2.240 PARAMETER *CASE* . . . 227
4.2.241 PARAMETER *BOM* . . . 228
4.2.242 PARAMETER *CHRMODE* . . . 228
4.2.242.1 CONSTANT *STOP* . . . 229
4.2.242.2 CONSTANT *IGNORE* . . . 229
4.2.242.3 CONSTANT *SUBSTITUTE* . . . 229
4.2.243 PARAMETER *SUBCHAR* . . . 230
4.2.244 PARAMETER *SYSTABLE* . . . 230
4.2.245 PARAMETER *USRTABLE* . . . 230
4.2.246 PARAMETER *REPORTFILE* . . . 233
4.2.247 OVERLAY *COMPRESS* . . . 234
4.2.248 OBJECT *FLAM4* . . . 235
4.2.249 OVERLAY *ENCRYPT* . . . 235
4.2.250 OBJECT *F4PWD* . . . 235
4.2.251 OBJECT *F4KME* . . . 236
4.2.252 OVERLAY *ENCODE* . . . 237
4.2.253 OBJECT *BASE64* . . . 237
4.2.254 OBJECT *ARMOR* . . . 239
4.2.255 OBJECT *BASE32* . . . 241
4.2.256 OBJECT *ARMOR* . . . 242
4.2.257 OBJECT *BASE16* . . . 244
4.2.258 OBJECT *ARMOR* . . . 246
4.2.259 OBJECT *OPGP* . . . 247
4.2.260 PARAMETER *LANG* . . . 249

4.2.261 PARAMETER *PLATFORM* . . . 249
4.2.262 OBJECT *HASH* . . . 250
4.2.263 OVERLAY *CHECK* . . . 251
4.2.264 OBJECT *FLAM4* . . . 252
4.2.265 OBJECT *SPACE* . . . 254
4.2.266 OVERLAY *SPLIT* . . . 255
4.2.267 PARAMETER *CCSID* . . . 255
4.2.268 PARAMETER *CASE* . . . 256
4.2.269 PARAMETER *BOM* . . . 256
4.2.270 PARAMETER *CHRMODE* . . . 257
4.2.270.1 CONSTANT *STOP* . . . 257
4.2.270.2 CONSTANT *IGNORE* . . . 257
4.2.270.3 CONSTANT *SUBSTITUTE* . . . 258
4.2.271 PARAMETER *SUBCHAR* . . . 258
4.2.272 PARAMETER *SYSTABLE* . . . 258
4.2.273 PARAMETER *USRTABLE* . . . 259
4.2.274 PARAMETER *REPORTFILE* . . . 261
4.2.275 PARAMETER *LANG* . . . 263
4.2.276 PARAMETER *PLATFORM* . . . 263
4.2.277 OBJECT *HASH* . . . 264
4.2.278 OVERLAY *CHECK* . . . 265
4.2.279 OBJECT *DIR* . . . 265
4.2.280 OVERLAY *LOGGING* . . . 266
4.2.281 OBJECT *STREAM* . . . 266
4.2.282 OBJECT *FILE* . . . 267
4.2.283 OBJECT *SYSTEM* . . . 267
4.2.284 OBJECT *MESSAGE* . . . 268
4.2.285 PARAMETER *INVERSE* . . . 269
4.3 COMMAND *XCNV* . . . 270
4.3.1 OBJECT *INPUT* . . . 271
4.3.2 OVERLAY *NET* . . . 271
4.3.3 OBJECT *IPN* . . . 271
4.3.4 OBJECT *IPS* . . . 272
4.3.5 OBJECT *MQS* . . . 272
4.3.6 OVERLAY *SAV* . . . 273
4.3.7 OBJECT *FILE* . . . 273
4.3.8 OVERLAY *FIO* . . . 273
4.3.9 OBJECT *BLK* . . . 274
4.3.10 PARAMETER *BLKSIZ* . . . 274

4.3.11 PARAMETER *PRNCTR* . . . 275
4.3.12 OBJECT *SUBSYS* . . . 275
4.3.13 OBJECT *REC* . . . 276
4.3.14 PARAMETER *PRNCTR* . . . 281
4.3.15 PARAMETER *BLKSIZ* . . . 281
4.3.16 OVERLAY *LENFMT* . . . 282
4.3.17 OBJECT *INTEGER* . . . 282
4.3.18 OBJECT *STRING* . . . 283
4.3.19 OBJECT *BCD* . . . 283
4.3.20 OBJECT *HOST* . . . 284
4.3.21 OBJECT *SUBSYS* . . . 284
4.3.22 OBJECT *TXT* . . . 285
4.3.23 OBJECT *SUBSYS* . . . 285
4.3.24 OBJECT *FL4* . . . 286
4.3.25 PARAMETER *PRNCTR* . . . 287
4.3.26 OVERLAY *CNV* . . . 287
4.3.27 OBJECT *BLK* . . . 287
4.3.28 OBJECT *REC* . . . 289
4.3.29 OBJECT *GZP* . . . 290
4.3.30 OBJECT *BZ2* . . . 291
4.3.31 OBJECT *LXZ* . . . 292
4.3.32 OBJECT *ZIP* . . . 292
4.3.33 OBJECT *CHR* . . . 293
4.3.34 PARAMETER *METHOD* . . . 294
4.3.35 PARAMETER *CASMOD* . . . 295
4.3.36 PARAMETER *FROM* . . . 296
4.3.37 PARAMETER *MODE* . . . 297
4.3.37.1 CONSTANT *STOP* . . . 297
4.3.37.2 CONSTANT *IGNORE* . . . 297
4.3.37.3 CONSTANT *SUBSTITUTE* . . . 298
4.3.37.4 CONSTANT *IDENTITY* . . . 298
4.3.38 PARAMETER *TO* . . . 298
4.3.39 PARAMETER *WRTBOM* . . . 299
4.3.40 PARAMETER *HDLBOM* . . . 299
4.3.41 PARAMETER *ENL2LF* . . . 299
4.3.42 PARAMETER *ELF2NL* . . . 300
4.3.43 PARAMETER *SUBCHR* . . . 300
4.3.44 PARAMETER *SYSTAB* . . . 300
4.3.45 PARAMETER *USRTAB* . . . 301

4.3.46 PARAMETER *REPFIL* . . . 304
4.3.47 PARAMETER *SKPBIN* . . . 305
4.3.48 PARAMETER *SKPEQU* . . . 305
4.3.49 OBJECT *BAS* . . . 305
4.3.50 OBJECT *HSH* . . . 307
4.3.51 OVERLAY *CHECK* . . . 308
4.3.52 OVERLAY *FMT* . . . 309
4.3.53 OBJECT *BIN* . . . 309
4.3.54 OBJECT *BLK* . . . 310
4.3.55 OBJECT *REC* . . . 311
4.3.56 OBJECT *TXT* . . . 312
4.3.57 OBJECT *XML* . . . 314
4.3.58 OBJECT *ENV* . . . 316
4.3.59 OBJECT *OUTPUT* . . . 316
4.3.60 OVERLAY *NET* . . . 316
4.3.61 OBJECT *IPN* . . . 316
4.3.62 OBJECT *IPS* . . . 317
4.3.63 OBJECT *MQS* . . . 317
4.3.64 OVERLAY *SAV* . . . 318
4.3.65 OBJECT *FILE* . . . 318
4.3.66 OVERLAY *FMT* . . . 318
4.3.67 OBJECT *BIN* . . . 319
4.3.68 OBJECT *BLK* . . . 319
4.3.69 OBJECT *REC* . . . 320
4.3.70 OBJECT *TXT* . . . 320
4.3.71 OBJECT *XML* . . . 322
4.3.72 OVERLAY *METHOD* . . . 323
4.3.73 OBJECT *STANDARD* . . . 323
4.3.74 OBJECT *PRETTYPRINT* . . . 323
4.3.75 OBJECT *MINIMIZED* . . . 324
4.3.76 OBJECT *DUMP* . . . 325
4.3.77 OVERLAY *CNV* . . . 327
4.3.78 OBJECT *BLK* . . . 327
4.3.79 OBJECT *REC* . . . 328
4.3.80 OBJECT *GZP* . . . 330
4.3.81 OBJECT *BZ2* . . . 331
4.3.82 OBJECT *LXZ* . . . 332
4.3.83 OBJECT *CHR* . . . 333
4.3.84 PARAMETER *METHOD* . . . 334

4.3.85 PARAMETER *CASMOD* . . . 335
4.3.86 PARAMETER *FROM* . . . 336
4.3.87 PARAMETER *MODE* . . . 337
4.3.87.1 CONSTANT *STOP* . . . 338
4.3.87.2 CONSTANT *IGNORE* . . . 338
4.3.87.3 CONSTANT *SUBSTITUTE* . . . 338
4.3.87.4 CONSTANT *IDENTITY* . . . 338
4.3.88 PARAMETER *TO* . . . 339
4.3.89 PARAMETER *WRTBOM* . . . 339
4.3.90 PARAMETER *HDLBOM* . . . 340
4.3.91 PARAMETER *ENL2LF* . . . 340
4.3.92 PARAMETER *ELF2NL* . . . 340
4.3.93 PARAMETER *SUBCHR* . . . 341
4.3.94 PARAMETER *SYSTAB* . . . 341
4.3.95 PARAMETER *USRTAB* . . . 341
4.3.96 PARAMETER *REPFIL* . . . 344
4.3.97 PARAMETER *SKPBIN* . . . 345
4.3.98 PARAMETER *SKPEQU* . . . 346
4.3.99 OBJECT *BAS* . . . 346
4.3.100 OBJECT *ARMOR* . . . 348
4.3.101 OBJECT *HSH* . . . 349
4.3.102 OVERLAY *CHECK* . . . 350
4.3.103 OVERLAY *FIO* . . . 351
4.3.104 OBJECT *BLK* . . . 351
4.3.105 OBJECT *FALLOC* . . . 352
4.3.106 OBJECT *SUBSYSTEM* . . . 354
4.3.107 OBJECT *SPACE* . . . 355
4.3.108 PARAMETER *PRNCTR* . . . 356
4.3.109 OBJECT *REC* . . . 356
4.3.110 OBJECT *FALLOC* . . . 359
4.3.111 OBJECT *SUBSYSTEM* . . . 362
4.3.112 OBJECT *SPACE* . . . 362
4.3.113 PARAMETER *PRNCTR* . . . 363
4.3.114 OVERLAY *LENFMT* . . . 364
4.3.115 OBJECT *INTEGER* . . . 364
4.3.116 OBJECT *STRING* . . . 365
4.3.117 OBJECT *BCD* . . . 365
4.3.118 OBJECT *HOST* . . . 366
4.3.119 OBJECT *TXT* . . . 366

4.3.120 OBJECT *FALLOC* . . . 367
4.3.121 OBJECT *SUBSYSTEM* . . . 369
4.3.122 OBJECT *SPACE* . . . 370
4.3.123 OBJECT *FL4* . . . 371
4.3.124 PARAMETER *ORGPRN* . . . 376
4.3.125 OBJECT *ENV* . . . 376
4.3.126 OBJECT *DIR* . . . 377
4.3.127 OBJECT *LOG* . . . 377
4.3.128 OBJECT *MESSAGE* . . . 378
4.3.129 OVERLAY *DESTINATION* . . . 379
4.3.130 OBJECT *STREAM* . . . 379
4.3.131 OBJECT *FILE* . . . 379
4.3.132 OBJECT *SYSTEM* . . . 380
4.4 COMMAND *ICNV* . . . 380
4.4.1 PARAMETER *IN* . . . 382
4.4.2 PARAMETER *OUT* . . . 382
4.4.3 OBJECT *FALLOC* . . . 382
4.4.4 OBJECT *SUBSYSTEM* . . . 385
4.4.5 OBJECT *SPACE* . . . 385
4.4.6 PARAMETER *RECORD* . . . 387
4.4.7 PARAMETER *METHOD* . . . 387
4.4.8 PARAMETER *CASE* . . . 388
4.4.9 PARAMETER *FROM* . . . 388
4.4.10 PARAMETER *MODE* . . . 390
4.4.10.1 CONSTANT *STOP* . . . 390
4.4.10.2 CONSTANT *IGNORE* . . . 390
4.4.10.3 CONSTANT *SUBSTITUTE* . . . 390
4.4.11 PARAMETER *BOM* . . . 391
4.4.12 PARAMETER *TO* . . . 391
4.4.13 PARAMETER *ENL2LF* . . . 392
4.4.14 PARAMETER *ELF2NL* . . . 392
4.4.15 PARAMETER *SUBCHR* . . . 393
4.4.16 PARAMETER *SYSTAB* . . . 393
4.4.17 PARAMETER *USRTAB* . . . 393
4.4.18 PARAMETER *REPORT* . . . 396
4.4.19 PARAMETER *APPEND* . . . 397
4.4.20 OBJECT *DIR* . . . 398
4.4.21 OBJECT *LOG* . . . 398
4.4.22 OBJECT *MESSAGE* . . . 399

4.4.23 OVERLAY *DESTINATION* . . . 400
4.4.24 OBJECT *STREAM* . . . 400
4.4.25 OBJECT *FILE* . . . 400
4.4.26 OBJECT *SYSTEM* . . . 401
4.5 COMMAND *XCHK* . . . 401
4.5.1 OBJECT *INPUT* . . . 402
4.5.2 OVERLAY *NET* . . . 402
4.5.3 OBJECT *IPN* . . . 403
4.5.4 OBJECT *IPS* . . . 403
4.5.5 OBJECT *MQS* . . . 404
4.5.6 OVERLAY *SAV* . . . 404
4.5.7 OBJECT *FILE* . . . 404
4.5.8 OVERLAY *FIO* . . . 404
4.5.9 OBJECT *BLK* . . . 405
4.5.10 PARAMETER *BLKSIZ* . . . 405
4.5.11 PARAMETER *PRNCTR* . . . 406
4.5.12 OBJECT *SUBSYS* . . . 406
4.5.13 OBJECT *REC* . . . 407
4.5.14 PARAMETER *PRNCTR* . . . 412
4.5.15 PARAMETER *BLKSIZ* . . . 412
4.5.16 OVERLAY *LENFMT* . . . 413
4.5.17 OBJECT *INTEGER* . . . 413
4.5.18 OBJECT *STRING* . . . 414
4.5.19 OBJECT *BCD* . . . 414
4.5.20 OBJECT *HOST* . . . 415
4.5.21 OBJECT *SUBSYS* . . . 415
4.5.22 OBJECT *TXT* . . . 416
4.5.23 OBJECT *SUBSYS* . . . 416
4.5.24 OBJECT *FL4* . . . 417
4.5.25 PARAMETER *PRNCTR* . . . 418
4.5.26 OVERLAY *CNV* . . . 418
4.5.27 OBJECT *BLK* . . . 418
4.5.28 OBJECT *REC* . . . 420
4.5.29 OBJECT *GZP* . . . 421
4.5.30 OBJECT *BZ2* . . . 422
4.5.31 OBJECT *LXZ* . . . 423
4.5.32 OBJECT *ZIP* . . . 423
4.5.33 OBJECT *CHR* . . . 424
4.5.34 PARAMETER *METHOD* . . . 425

4.5.35 PARAMETER *CASMOD* . . . 426
4.5.36 PARAMETER *FROM* . . . 427
4.5.37 PARAMETER *MODE* . . . 428
4.5.37.1 CONSTANT *STOP* . . . 428
4.5.37.2 CONSTANT *IGNORE* . . . 428
4.5.37.3 CONSTANT *SUBSTITUTE* . . . 429
4.5.37.4 CONSTANT *IDENTITY* . . . 429
4.5.38 PARAMETER *TO* . . . 429
4.5.39 PARAMETER *WRTBOM* . . . 430
4.5.40 PARAMETER *HDLBOM* . . . 430
4.5.41 PARAMETER *ENL2LF* . . . 430
4.5.42 PARAMETER *ELF2NL* . . . 431
4.5.43 PARAMETER *SUBCHR* . . . 431
4.5.44 PARAMETER *SYSTAB* . . . 431
4.5.45 PARAMETER *USRTAB* . . . 432
4.5.46 PARAMETER *REPFIL* . . . 435
4.5.47 PARAMETER *SKPBIN* . . . 436
4.5.48 PARAMETER *SKPEQU* . . . 436
4.5.49 OBJECT *BAS* . . . 436
4.5.50 OBJECT *HSH* . . . 438
4.5.51 OVERLAY *CHECK* . . . 439
4.5.52 OVERLAY *FMT* . . . 440
4.5.53 OBJECT *BIN* . . . 440
4.5.54 OBJECT *BLK* . . . 441
4.5.55 OBJECT *REC* . . . 442
4.5.56 OBJECT *TXT* . . . 443
4.5.57 OBJECT *XML* . . . 445
4.5.58 OBJECT *ENV* . . . 447
4.5.59 OBJECT *DIR* . . . 447
4.5.60 OBJECT *LOG* . . . 448
4.5.61 OBJECT *MESSAGE* . . . 448
4.5.62 OVERLAY *DESTINATION* . . . 449
4.5.63 OBJECT *STREAM* . . . 449
4.5.64 OBJECT *FILE* . . . 450
4.5.65 OBJECT *SYSTEM* . . . 450
4.6 COMMAND *HASH* . . . 451
4.6.1 OBJECT *DIR* . . . 452
4.6.2 OBJECT *LOG* . . . 453
4.6.3 OBJECT *MESSAGE* . . . 453

4.6.4 OVERLAY *DESTINATION* . . . 455
4.6.5 OBJECT *STREAM* . . . 455
4.6.6 OBJECT *FILE* . . . 455
4.6.7 OBJECT *SYSTEM* . . . 456
4.6.8 OVERLAY *CHECK* . . . 456
4.7 COMMAND *DIFF* . . . 457
4.7.1 OVERLAY *READ* . . . 459
4.7.2 OBJECT *BINARY* . . . 460
4.7.3 PARAMETER *BLKSIZE* . . . 461
4.7.4 OVERLAY *DECRYPT* . . . 461
4.7.5 OBJECT *F4PWD* . . . 461
4.7.6 OBJECT *F4KME* . . . 462
4.7.7 OVERLAY *FDECODE* . . . 463
4.7.8 PARAMETER *PRNCONTROL* . . . 463
4.7.9 OBJECT *SUBSYSTEM* . . . 464
4.7.10 PARAMETER *PLATFORM* . . . 464
4.7.11 OBJECT *HASH* . . . 465
4.7.12 OVERLAY *CHECK* . . . 466
4.7.13 PARAMETER *CHECK* . . . 467
4.7.14 OBJECT *CHARACTER* . . . 467
4.7.15 PARAMETER *BLKSIZE* . . . 468
4.7.16 PARAMETER *CCSID* . . . 468
4.7.17 PARAMETER *BOM* . . . 470
4.7.18 PARAMETER *ENL2LF* . . . 470
4.7.19 OVERLAY *DECRYPT* . . . 470
4.7.20 OBJECT *F4PWD* . . . 471
4.7.21 OBJECT *F4KME* . . . 471
4.7.22 PARAMETER *PRNCONTROL* . . . 472
4.7.23 OBJECT *SUBSYSTEM* . . . 472
4.7.24 PARAMETER *LANG* . . . 473
4.7.25 PARAMETER *PLATFORM* . . . 473
4.7.26 OBJECT *HASH* . . . 474
4.7.27 OVERLAY *CHECK* . . . 475
4.7.28 PARAMETER *CHECK* . . . 476
4.7.29 OBJECT *TEXT* . . . 476
4.7.30 PARAMETER *BLKSIZE* . . . 478
4.7.31 PARAMETER *CCSID* . . . 478
4.7.32 PARAMETER *BOM* . . . 479
4.7.33 PARAMETER *ENL2LF* . . . 480

4.7.34 OVERLAY *DECRYPT* . . . 480
4.7.35 OBJECT *F4PWD* . . . 480
4.7.36 OBJECT *F4KME* . . . 481
4.7.37 PARAMETER *PRNCONTROL* . . . 482
4.7.38 OBJECT *SUBSYSTEM* . . . 482
4.7.39 PARAMETER *LANG* . . . 483
4.7.40 PARAMETER *PLATFORM* . . . 483
4.7.41 OBJECT *HASH* . . . 484
4.7.42 OVERLAY *CHECK* . . . 485
4.7.43 PARAMETER *CHECK* . . . 485
4.7.44 OBJECT *XML* . . . 486
4.7.45 PARAMETER *BLKSIZE* . . . 487
4.7.46 PARAMETER *CCSID* . . . 487
4.7.47 PARAMETER *BOM* . . . 488
4.7.48 PARAMETER *ENL2LF* . . . 489
4.7.49 OVERLAY *DECRYPT* . . . 489
4.7.50 OBJECT *F4PWD* . . . 489
4.7.51 OBJECT *F4KME* . . . 490
4.7.52 OBJECT *SUBSYSTEM* . . . 491
4.7.53 PARAMETER *LANG* . . . 491
4.7.54 PARAMETER *PLATFORM* . . . 492
4.7.55 OBJECT *HASH* . . . 492
4.7.56 OVERLAY *CHECK* . . . 493
4.7.57 PARAMETER *CHECK* . . . 494
4.7.58 OBJECT *RECORD* . . . 494
4.7.59 PARAMETER *PRNCONTROL* . . . 498
4.7.60 PARAMETER *BLKSIZE* . . . 499
4.7.61 OVERLAY *LENFORMAT* . . . 499
4.7.62 OBJECT *INTEGER* . . . 500
4.7.63 OBJECT *STRING* . . . 500
4.7.64 OBJECT *BCD* . . . 501
4.7.65 OBJECT *HOST* . . . 501
4.7.66 OVERLAY *DECRYPT* . . . 502
4.7.67 OBJECT *F4PWD* . . . 502
4.7.68 OBJECT *F4KME* . . . 503
4.7.69 OVERLAY *FDECODE* . . . 503
4.7.70 PARAMETER *CCSID* . . . 504
4.7.71 PARAMETER *BOM* . . . 505
4.7.72 OBJECT *SUBSYSTEM* . . . 505

4.7.73 PARAMETER *LANG* . . . 506
4.7.74 PARAMETER *PLATFORM* . . . 506
4.7.75 OBJECT *HASH* . . . 507
4.7.76 OVERLAY *CHECK* . . . 508
4.7.77 PARAMETER *CHECK* . . . 509
4.7.78 OBJECT *FLAM4* . . . 509
4.7.79 PARAMETER *PRNCONTROL* . . . 510
4.7.80 PARAMETER *CCSID* . . . 510
4.7.81 PARAMETER *BOM* . . . 512
4.7.82 PARAMETER *LANG* . . . 512
4.7.83 PARAMETER *PLATFORM* . . . 512
4.7.84 OBJECT *HASH* . . . 513
4.7.85 OVERLAY *CHECK* . . . 514
4.7.86 OBJECT *AUTO* . . . 515
4.7.87 PARAMETER *BLKSIZE* . . . 516
4.7.88 PARAMETER *CCSID* . . . 516
4.7.89 PARAMETER *BOM* . . . 517
4.7.90 PARAMETER *ENL2LF* . . . 518
4.7.91 OVERLAY *DECRYPT* . . . 518
4.7.92 OBJECT *F4PWD* . . . 518
4.7.93 OBJECT *F4KME* . . . 519
4.7.94 PARAMETER *LANG* . . . 520
4.7.95 PARAMETER *PLATFORM* . . . 520
4.7.96 OBJECT *HASH* . . . 521
4.7.97 OVERLAY *CHECK* . . . 522
4.7.98 PARAMETER *CHECK* . . . 522
4.7.99 PARAMETER *FILE* . . . 523
4.7.100 OVERLAY *COMPARE* . . . 523
4.7.101 OBJECT *BINARY* . . . 523
4.7.102 PARAMETER *BLKSIZE* . . . 524
4.7.103 OVERLAY *DECRYPT* . . . 525
4.7.104 OBJECT *F4PWD* . . . 525
4.7.105 OBJECT *F4KME* . . . 526
4.7.106 OVERLAY *FDECODE* . . . 526
4.7.107 PARAMETER *PRNCONTROL* . . . 527
4.7.108 OBJECT *SUBSYSTEM* . . . 528
4.7.109 PARAMETER *PLATFORM* . . . 528
4.7.110 OBJECT *HASH* . . . 529
4.7.111 OVERLAY *CHECK* . . . 530

4.7.112 PARAMETER *CHECK* . . . 530
4.7.113 OBJECT *CHARACTER* . . . 531
4.7.114 PARAMETER *BLKSIZE* . . . 532
4.7.115 PARAMETER *CCSID* . . . 532
4.7.116 PARAMETER *BOM* . . . 533
4.7.117 PARAMETER *ENL2LF* . . . 534
4.7.118 OVERLAY *DECRYPT* . . . 534
4.7.119 OBJECT *F4PWD* . . . 534
4.7.120 OBJECT *F4KME* . . . 535
4.7.121 PARAMETER *PRNCONTROL* . . . 536
4.7.122 OBJECT *SUBSYSTEM* . . . 536
4.7.123 PARAMETER *LANG* . . . 537
4.7.124 PARAMETER *PLATFORM* . . . 537
4.7.125 OBJECT *HASH* . . . 538
4.7.126 OVERLAY *CHECK* . . . 539
4.7.127 PARAMETER *CHECK* . . . 539
4.7.128 OBJECT *TEXT* . . . 540
4.7.129 PARAMETER *BLKSIZE* . . . 541
4.7.130 PARAMETER *CCSID* . . . 542
4.7.131 PARAMETER *BOM* . . . 543
4.7.132 PARAMETER *ENL2LF* . . . 543
4.7.133 OVERLAY *DECRYPT* . . . 544
4.7.134 OBJECT *F4PWD* . . . 544
4.7.135 OBJECT *F4KME* . . . 545
4.7.136 PARAMETER *PRNCONTROL* . . . 545
4.7.137 OBJECT *SUBSYSTEM* . . . 546
4.7.138 PARAMETER *LANG* . . . 546
4.7.139 PARAMETER *PLATFORM* . . . 547
4.7.140 OBJECT *HASH* . . . 547
4.7.141 OVERLAY *CHECK* . . . 548
4.7.142 PARAMETER *CHECK* . . . 549
4.7.143 OBJECT *XML* . . . 549
4.7.144 PARAMETER *BLKSIZE* . . . 550
4.7.145 PARAMETER *CCSID* . . . 551
4.7.146 PARAMETER *BOM* . . . 552
4.7.147 PARAMETER *ENL2LF* . . . 552
4.7.148 OVERLAY *DECRYPT* . . . 553
4.7.149 OBJECT *F4PWD* . . . 553
4.7.150 OBJECT *F4KME* . . . 554

4.7.151 OBJECT *SUBSYSTEM* . . . 554
4.7.152 PARAMETER *LANG* . . . 555
4.7.153 PARAMETER *PLATFORM* . . . 555
4.7.154 OBJECT *HASH* . . . 556
4.7.155 OVERLAY *CHECK* . . . 557
4.7.156 PARAMETER *CHECK* . . . 557
4.7.157 OBJECT *RECORD* . . . 558
4.7.158 PARAMETER *PRNCONTROL* . . . 562
4.7.159 PARAMETER *BLKSIZE* . . . 562
4.7.160 OVERLAY *LENFORMAT* . . . 563
4.7.161 OBJECT *INTEGER* . . . 563
4.7.162 OBJECT *STRING* . . . 564
4.7.163 OBJECT *BCD* . . . 564
4.7.164 OBJECT *HOST* . . . 565
4.7.165 OVERLAY *DECRYPT* . . . 565
4.7.166 OBJECT *F4PWD* . . . 565
4.7.167 OBJECT *F4KME* . . . 566
4.7.168 OVERLAY *FDECODE* . . . 567
4.7.169 PARAMETER *CCSID* . . . 567
4.7.170 PARAMETER *BOM* . . . 568
4.7.171 OBJECT *SUBSYSTEM* . . . 569
4.7.172 PARAMETER *LANG* . . . 569
4.7.173 PARAMETER *PLATFORM* . . . 570
4.7.174 OBJECT *HASH* . . . 570
4.7.175 OVERLAY *CHECK* . . . 571
4.7.176 PARAMETER *CHECK* . . . 572
4.7.177 OBJECT *FLAM4* . . . 572
4.7.178 PARAMETER *PRNCONTROL* . . . 573
4.7.179 PARAMETER *CCSID* . . . 574
4.7.180 PARAMETER *BOM* . . . 575
4.7.181 PARAMETER *LANG* . . . 575
4.7.182 PARAMETER *PLATFORM* . . . 575
4.7.183 OBJECT *HASH* . . . 576
4.7.184 OVERLAY *CHECK* . . . 577
4.7.185 OBJECT *AUTO* . . . 578
4.7.186 PARAMETER *BLKSIZE* . . . 579
4.7.187 PARAMETER *CCSID* . . . 579
4.7.188 PARAMETER *BOM* . . . 581
4.7.189 PARAMETER *ENL2LF* . . . 581

4.7.190 OVERLAY *DECRYPT* . . . 581
4.7.191 OBJECT *F4PWD* . . . 582
4.7.192 OBJECT *F4KME* . . . 582
4.7.193 PARAMETER *LANG* . . . 583
4.7.194 PARAMETER *PLATFORM* . . . 583
4.7.195 OBJECT *HASH* . . . 584
4.7.196 OVERLAY *CHECK* . . . 585
4.7.197 PARAMETER *CHECK* . . . 585
4.7.198 PARAMETER *FILE* . . . 586
4.7.199 OVERLAY *LOGGING* . . . 586
4.7.200 OBJECT *STREAM* . . . 586
4.7.201 OBJECT *FILE* . . . 587
4.7.202 OBJECT *SYSTEM* . . . 587
4.7.203 OBJECT *MESSAGE* . . . 588
4.8 COMMAND *UTIL* . . . 589
4.8.1 OVERLAY *RUN* . . . 590
4.8.2 OBJECT *DIR* . . . 590
4.8.3 OBJECT *LOG* . . . 591
4.8.4 OBJECT *MESSAGE* . . . 591
4.8.5 OVERLAY *DESTINATION* . . . 592
4.8.6 OBJECT *STREAM* . . . 593
4.8.7 OBJECT *FILE* . . . 593
4.8.8 OBJECT *SYSTEM* . . . 594

5 Available built-in functions **595**
5.1 FUNCTION *SYNTAX* . . . 595
5.2 FUNCTION *HELP* . . . 596
5.3 FUNCTION *MANPAGE* . . . 597
5.4 FUNCTION *GENDOCU* . . . 597
5.5 FUNCTION *GENPROP* . . . 598
5.6 FUNCTION *SETPROP* . . . 598
5.7 FUNCTION *CHGPROP* . . . 599
5.8 FUNCTION *DELPROP* . . . 599
5.9 FUNCTION *GETPROP* . . . 600
5.10 FUNCTION *SETOWNER* . . . 601
5.11 FUNCTION *GETOWNER* . . . 601
5.12 FUNCTION *SETENV* . . . 601
5.13 FUNCTION *GETENV* . . . 602
5.14 FUNCTION *DELENV* . . . 602

5.15 FUNCTION *TRACE* . . . 602
5.16 FUNCTION *CONFIG* . . . 603
5.17 FUNCTION *GRAMMAR* . . . 603
5.18 FUNCTION *LEXEM* . . . 603
5.19 FUNCTION *LICENSE* . . . 604
5.20 FUNCTION *VERSION* . . . 604
5.21 FUNCTION *ABOUT* . . . 604
5.22 FUNCTION *ERRORS* . . . 605

6 GLOSSARY **606**

A LEXEM **613**

B GRAMMAR **615**

C PROPERTIES **617**

D RETURN CODES **745**
D.1 Special condition codes . . . 746

E REASON CODES **747**

F VERSION **751**

G ABOUT **753**

7 Index **762**

Frankenstein Limes Command Line (FLCL®) Copyright © limes datentechnik ® gmbh All rights reserved

Trademarks Below, you can find all trademarks or registered trademarks of limes datentechnik ® gmbh. These trademarked terms are marked with the appropriate symbol (® or ™), indicating registered or common law trademarks owned by limes datentechnik ® gmbh at the time this information was published. The following terms are trademarks of limes datentechnik ® gmbh in Germany, other countries, or both:

- limes® - Short company name of the owner of this document
- limes datentechnik® - Company name of the owner of this document
- FLCL® - Frankenstein Limes Command Line
- FLCC® - Frankenstein Limes Control Center
- FLAM® - Frankenstein Limes Access Method
- FLUC® - Frankenstein Limes Universal Converter
- FLIES® - Frankenstein Limes Integrated Extended Security
- FLAMFILE® - A file based on FLAM syntax

PREFACE

The FLCL is based on the CLE/P library. The CLE/P library was developed by limes datentechnik and released as open source under the ZLIB license. CLE/P is a compiler to provide a platform-independent command line interface for each kind of batch processing environment.

The CLE/P library provides a lot of features including all the built-in functions below. For example: We use this library to automatically create this document as part of our build process; If we add a parameter to the CLE/P tables and build the FL5 project, this manual will be regenerated as well in order to be always up to date.

This manual is generated with the built-in function GENDOCU provided by CLE/P library by calling the command below:

```
flcl gendocu flclbook.txt
```

Chapter 1

Abstract

This document provides information about using the Frankenstein Limes Command Line (FLCL). It contains advanced guidelines and information to run the commands provided by this batch interface. FLCL is the main utility to run all the different subprograms of the FL5 infrastructure.

Beside FLCL, the FL5 infrastructure consists of the components below:

- FLUCUP - FLUC Subprogram as C interface (DLL)
  - Data source to target conversion as described in this document
  - FLCL command line = parameter string for subprogram
  - `ret=flucinfo("get.file=test.txt");`
- FLUCUPLB - FLUC Subprogram as load module interface
  - All functions of FLUCUP (flucconv) are provided as load modules for simple integration in COBOL, PL1 or assembler programs.
  - `FCUCONV("read.file=DD:INPUT")`
- FLCBYT - Byte interface for original data (C like DLL)
  - You can transparently read and write each kind of data source or target as byte stream or records with a C like file-I/O interface using the read or write overlays of the FLCL-CONV/XCNV commands described in this document.
  - `f=fcbopen("read.file=text.txt","format.record()");`
  - `f=fcbopen("write.text(file=out.txt comp.bzip())","format.text()");`
- FLCBYTBF - C++ streambuf class for I/O streams based on FLCBYT
  - You can pass an flcbyt_buffer object to an istream or ostream instance in C++ and use the byte interface as a regular stream.
  - `flcbyt_buffer buf("read.file=text.txt","format.bin()");std::istream in(&buf);`
  - `flcbyt_buffer buf("write.text(file=out.txt)","format.text()");std::ostream out(&buf);`
- FLCBYTJava - Java Native Interface (JNI) library for the FLUC byte interface and corresponding stream classes for Java (like FLCBYTBF for C++)
  - This interface can be used to read or write FLAMFILEs or other encoded, encrypted, compressed or clear files through Input/OutputStreams in Java.
- FLCRECLB - Record interface for original data (load modules)

  - You can transparently read (GET) and write (PUT) each kind of data source or target in a record-I/O-like interface also using the read and write unions of the CONV command or the input and output objects of the XCNV command in the corresponding file open function. All functions are provided as separate load module for simple usage in COBOL, PL1 or assembler programs.
  - `FCROPN("read.auto(file=<SYSUID>.VASM.KSDS ccsid=1141)")`
  - `FCROPN("write.text(method=win, ccsid=1252, comp.gzip(level=9))")`
- FLCICV - FLUC character conversion module (ICONV like DLL)
  - You can transparently use this libiconv-compatible interface for memory to memory character conversions. It supports all feature of our character conversion component (CNVCHR), including:
    * Stop, ignore, substitution and transliteration for invalid characters
    * EBCDIC new line to line feed management
    * User table, subset support (SEPA, String.Latin(XOEF/NPA), ...)
    * UTF-8 with 5 and 6 byte long encodings
    * Case mapping, BOM management (including byte order change support)
    * Comprehensive reporting of error positions (SUB, IGN, XLT, ...)
  - `h=fliconv_open("IBM1141//ENL2LF","UTF-16LE//BOM//IGNORE//TRANSLIT(ICONV)//USRTAB(DELA)//REPORT(DD:REPORT)");`
  - `fliconv(h,&indat,&inlen,&outdat,&outlen);`
  - `fliconv_close(h);`
  - Look for FLICONV utility program and the corresponding sample source

This document is also useful for users of the byte, record, element or subprogram interfaces of the FL5 infrastructure, because all these interfaces use the same syntax as the command line.

Various C/C++ sample programs for the APIs above can be found in the library SRCLIBC as part of the installation package for mainframe systems. The corresponding compile and link step is in JOBLIB(SBUILD).

Additional COBOL, PL1 and Assembler samples can be found in the library SRCLIB. The corresponding compile and link procedures are also located in JOBLIB(SBUILD) as separate steps.

For other platforms (Windows, UNIX) the C, C++ and Java sample program sources are located in the *sample* directory and the compile and link procedures can be found in the Makefile of the same directory.

Beside all these new FL5 based components, the following FLAM4 components are still supported:

- FLAM4-Utility - The command line program, also available as command FLAM in FLCL.
- FLAM4-Subprogram - Integration of file to file compression and decompression in other programs (works like FLUCUP[LB]. For new projects we recommend to use the FLUCUP[LB]).
- FLAM4-Record interface - Record-wise (length and data) read (GET) and write (PUT) access to compressed FLAMFILES.
- FLAM4-Subsystem (only on z/OS) - Application-transparent integration of FLAM at SVC99 file allocation
  - `FILE DD DSN=&SYSUID.TEST,DISP=...,...,SUBSYS=FLAM`

With FLAM5 new CLIST, PANELS and MESSAGES are added for ISPF on z/OS:

- FLINFO - ISPF line command to show determined file attributes
- FLVIEW - ISPF line command to show content of different files
  - normal host data sets, members of PDS/PDSE, VSAM data sets
  - Directory content of PDS/PDSE, FLAMFILES, GZIP/BZIP/XZ files
  - binary transfered XML or text files from other platforms

  - base encoded, encrypted, compressed files
  - and much more
- FLTEXT - ISPF line command like FLVIEW, but it produce dumps for binary files
- FLVEDIT- ISPF line command to edit content of different files
  - works like FLVIEW, but you can edit the data
  - the data is then written back in the original format
  - for example you can edit UTF-8 XML data in a GZIP file over ISPF
- FLHLP - ISPF command for interactive help of FLCL commands
- FLSYN - ISPF command for interactive syntax of FLCL commands
- FLMAN - ISPF command for interactive manual pages of FLCL commands
- FLDOC - ISPF command for documentation generation of FLCL
- FLGRA - ISPF command to show grammar of FLCL commands
- FLLEX - ISPF command to show lexems of FLCL commands
- FLABO - ISPF command to show about information of FLCL
- FLVSN - ISPF command to show version information of FLCL
- FLLIC - ISPF command to show license information of FLCL

Chapter 2

COMMAND LINE PROCESSOR

This command line parser provides a list of commands and some built-in functions, please use *MANPAGE*, *HELP* and *SYNTAX* to get extensive information about these capabilities.

Additional You can use the build-in function *GENDOCU* to generate a complete or parts of the user manual.

To read the parameter of a command, a compiler (command line processor CLP) is applied. To see the regular expressions (lexems) and the corresponding grammar, please use the build in functions *LEXEM* and *GRAMMAR*.

The return/condition/exit codes of the executable and the reason codes of the different commands can be reviewed with the built-in function *ERRORS*. See Appendix D and, if available, Appendix E for the meaning of the used return and reason codes.

The command line executer (CLE) uses an owner management in order to separate the settings for different clients and a property management for each command. If problems occur, you can activate a trace and manage all configuration settings.

For each command execution you can define the owner and environment variables which are managed over the configuration file.

The default trace file is *stdout*. If you activate the trace before a trace file is defined, the trace will be printed on the screen.

Last but not least you can determine license, version and other information about the program.

This is the RELEASE build version of the FLAMCLEP.

2.1 USED ENVIRONMENT VARIABLES

- LANG - to determine the CCSID on EBCDIC systems
- HOME - to determine the home directory on UNIX/WIN
- USER - to determine the current user id on UNIX/WIN
- OWNERID - used for current owner if not already defined
- FLCL_CONFIG_FILE - the configuration filename (default is *$HOME/.flcl.config* on UNIX/WIN or *&SYSUID..FLCL.CONFIG* on mainframes)
- FLCL_DEFAULT_OWNER_ID - the default owner ID (default is *de.limes*)
- owner_FLCL_command_PROPERTY_FILENAME - To overrule default property file names
- path_argument - to overrule the hard coded default property value

Environment variables can be set from the system or in the program configuration file with the build-in function *SETENV*. Especially on mainframe systems the configuration file is an easy way to define environment variables. Additional on host the DD name STDENV is supported. The DD:STDENV allows you to define the FLCL_CONFIG_FILE, FLCL_DEFAULT_OWNER_ID, LANG and other environment variables for example direct in your JCL:

```
//STDENV   DD *
FLCL_CONFIG_FILE=GLOBAL.FLCL.CONFIG
LANG=de_DE.IBM-1141
HOME=/u/hugo
USER=hugo
/*
```

Often it will be useful to have a dedicated environment per user on mainframes. In such case it makes sense to define the environment in a dedicated file for each user.

```
//STDENV   DD DSN=USER.ENVIR(&SYSUID.), DSP=SHR
```

Beside all the environment variables managed by CLE you can also set all properties as environment variables to overrule the hard coded default values with CLP. If a property defined as environment variable and over a property file, then the value in the property file overrules the settings over the environment. The environment variable name for each property are builded by the rules below:

- convert all letters to upper case
- replace all dots (`.`) by underline (_)

To get a list and help for all properties pleas use the built-in function *GENPROP* to generate property files. The properties can be defined per owner, per program and general. The owner specific definition overrules the program specific definition and the program specific definition overrules the general definition. Examples:

```
CONV_READ_TEXT_ENL2LF=OFF #in general the 0x15 to 0x25 conversion is off#
HUGO_FLCL_CONV_READ_TEXT_ENL2LF=ON # for owner 'hugo' the conversion is on#
```

The value string behind the sign (including the comment) will be used as supplement for the command line processor. Aliases are not supported in this case. You can only define properties for the main argument. If a string must be enclosed with apostrophe, please don't use double quotation marks, because these are used additional if a new property file build based on the environment settings.

```
FLCL_ICNV_FROM='IBM-1141'          # this is the best solution
FLCL_ICNV_TO=UTF-8                 # "UTF-8" could result in errors
```

See Appendix C for the current property file content.

2.2 FILENAME MAPPING

All filenames used by CLEP are additionally mapped based on the rules below:

- The tilde character (~) is replaced with the string "<HOME>"
- Each value enclosed with angle brackets (<>) are replaced with the corresponding environment variable (`<OWNERID>`). If the environment variable not defined the replacements below are still possible:
  - <SYSUID> - Current user id in upper case
  - <USER> - Current user id (case sensitive)
  - <CUSER> - Current user id in upper case == <SYSUID>
  - <Cuser> - Current user id in title case
  - <cuser> - Current user id in lower case
  - <HOME> - Replaces with the users data directory, if this not available the replacements below are done:
    - On UNIX with /home/<USER>

    * On USS with /u/<USER>
    * On ZOS with <SYSUID>

- DD names on mainframes must be prefixed with "DD:"
- Data set names on mainframes are always full qualified
- Path names on mainframes must contain a least one slash (/)
- Data set names on USS must start with //
  - Full qualified names with HLQ must enclose in apostrophes ("//'")
  - If apostrophes are not used the SYSUID is prefixed as HLQ
- Normal file names on other platforms could be relative

ATTENTION: If a requested environment variable not defined, the replacement is done with a empty string. This could result in a unexpected behavior.

To use a "<" or "~" as a part of a filename the character must be specified twice.

Beside this rules you can also use the replacement technologies of your shell, but on some platforms $HOME, $USER or something like this are not available, for such cases the possibilities above are implemented.

This file name mapping is provided over the library CLEPUTL and should also be used for file names managed by the commands supported with this program.

2.3 SPECIAL EBCDIC CODE PAGE SUPPORT

To interpret commands correctly the punctuation characters below are on different code points depending on the EBCDIC CCSID used to enter these values.

```
CRITICAL PUNCTUATION CHARACTERS: ! $ # @ [ \ ] ^ ` { | } ~
```

These critical characters are interpreted dependent on the environment variable LANG. If the environment variable LANG is not defined then the compilation default CCSID (f.e. IBM-1047 on USS and IBM1141 on ZOS) is used. Below you can find the current list of supported CCSIDs on EBCDIC systems.

```
SUPPORTED EBCDIC CODE PAGES FOR COMMAND ENTRY:
   "IBM-1140","IBM-1141","IBM-1142","IBM-1143",
   "IBM-1144","IBM-1145","IBM-1146","IBM-1147",
   "IBM-1148","IBM-1149","IBM-1153","IBM-1154",
   "IBM-1156","IBM-1122","IBM-1047"
   "IBM-500","IBM-273","IBM-037","IBM-875","IBM-424"
```

You can define the code page explicit (LANG=de_DE.IBM-1141) or only the language code (LANG=de_DE, LANG=C). If only the language code defined then the CCSID is derived from the language code (DE=IBM-1141, US=IBM-1047, C=IBM-1047, ...).

If it possible these critical characters are also converted for print outs. At output it is not possible to convert anything correctly, because some strings for print out are coming from other sources (like system messages and others). Only all known literals are converted, for unknown variables such a conversion is not possible and CLEP expect that such strings are encoded in the correct system code page, but we can not guaranty this.

On ASCII/UTF-8 platforms a miss interpretation of punctuation characters smaller then 128 is not possible. On such platforms the LANG variable is not used for command interpretation or printouts.

Chapter 3

PROGRAM *flcl*

3.1 SYNOPSIS

```
HELP:   Frankenstein Limes(R) Command Line for FLUC, FLAM and FLIES
PATH:   LIM
TYPE:   PROGRAM
SYNTAX: :> flcl COMMAND/FUNCTION ...
```

3.2 DESCRIPTION

Frankenstein Limes Command Line (FLCL) is a batch utility to execute the following subprograms:

- FLUC - Frankenstein Limes Universal Converter (conversion)
- FLAM - Frankenstein Limes Access Method (compression and encryption)
- FLIES - Frankenstein Limes Integrated Extended Security (management)

Each subprogram is also available as DLL/SO or load modules to integrate FLAM into your own applications. The subprogram interface supports each command as dedicated function, which accepts the corresponding command string.

For example this call of FLCL:

```
   :> flcl info "get.file=test.txt"
```

is also available in C programming language as:

```
   ret=flucinfo("get.file=test.txt");
```

The subprograms are called through the commands below:

- INFO - a command of the subprogram *FLUC* → flucinfo()
- CONV - a command of the subprogram *FLUC* → flucconv()
- XCNV - a command of the subprogram *FLUC* → flucxcnv()
- XCHK - a command of the subprogram *FLUC* → flucxchk()
- HASH - a command of the subprogram *FLUC* → fluchash()
- UTIL - a command of the subprogram *FLUC* → flucutil()

- COMP - a command of the subprogram *FLAM* → flamcomp()
- DECO - a command of the subprogram *FLAM* → flamdeco()
- FIND - a command of the subprogram *FLIES* → flisfind()
- CHNG - a command of the subprogram *FLIES* → flischng()

The FLCL can be used in scripts to automate batch processing. The command interpretation is platform independent and optimized for different shells like Windows® batch files, Unix bash or z/OS® JCL.

3.2.1 USED ENVIRONMENT VARIABLES

- LANG - the default language and CCSID/encoding string for this platform (Format: de_DE.UTF-8 or de_DE.IBM-1141)
- FL_PLATFORM Platform definition (WIN/UNX/MAC/ZOS/USS/VSE/BS2) for emulation at read or write operations
- FL_DEFAULT_ASCII_CCSID - default CCSID for ASCII character sets (if ASCII machine and LANG, then default is taken from LANG else default is *ISO-8859-1*))
- FL_DEFAULT_EBCDIC_CCSID - default CCSID for EBCDIC character sets (if EBCDIC machine and LANG, then default is taken from LANG else default is *IBM-1148*))
- FL_PRIMARY_SPACE_TRK - default primary space in tracks (only relevant on mainframes, default is 600)
- FL_SECONDARY_SPACE_TRK - default secondary space in tracks (only relevant on mainframes, default is 1200)
- FL_DIRECTORY_BLOCKS - default directory blocks (only relevant on mainframes, default is 45)
- FL_XZ_EXPANSION_FACTOR - expansion factor for XZ decompression (default is 10)
- FL_XZ_COMPRESSION_FACTOR - compression factor for XZ compression (default is 2)
- FL_GZ_EXPANSION_FACTOR - expansion factor for GZIP decompression (default is 5)
- FL_GZ_COMPRESSION_FACTOR - compression factor for GZIP compression (default is 2)
- FL_BZ_EXPANSION_FACTOR - expansion factor for BZIP2 decompression (default is 6)
- FL_BZ_COMPRESSION_FACTOR - compression factor for BZIP2 compression (default is 2)
- FL_RECORD_FORMAT_MAPPING=DATATYPE - binary data as binary (RECF=BIN), text as text (RECF=TXT) if this variable is not set (DEFAULT), records with length field (RECF=VAR) are used

All these environment variables can be managed through the built-in functions SETENV, GETENV and DELENV within the FLCL configuration file. On mainframes, you can also use the DD:STDENV to define environment variables. The environment variable LANG is also used on EBCDIC systems to interpret several syntax characters correctly, which depends on the codepage. It is important to set this variable to the correct value on EBCDIC systems. If this environment variable is not set, literals are interpreted in IBM-1141 on ZOS and in IBM-1047 on USS. If you use another CCSID for entry, you must define the environment variable LANG to enable correct parsing for strings that contain these special characters.

see *USED ENVIRONMENTS VARIABLES* and *SPECIAL EBCDIC CODE PAGE SUPPORT* of Command Line Processor

3.2.2 FILENAME HANDLING

The filename handling must be split in 2 parts. One part reflects files needed by the built-in functions. This definition of filenames depends on the platform, shell and environment used to run FLCL. FLCL uses the CLEPUTL library for filename mapping, which means that all replacement mechanisms described for CLEP are also valid for each filename used within a command.

see *FLENAME HANDLING* of Command Line Processor

All filenames specified inside of command strings are parsed and handled in a more unified way across the different platforms. Standard replacements by shells are mainly not usable inside of the command syntax.

The filename handling works as follows:

For mainframe systems (POSIX(OFF)):

Usually, a filename has to be defined fully qualified, because a dynamic allocation is done in most cases. To simplify such a definition, the following replacements can be used:

```
<envar>  - to expand an environment variable
```

If the environment variables below are not defined, the following defaults are used:

```
<SYSUID> - The current user id (Example: file='<SYSUID>.TEST.DATA')
<USER>   - The current user id (Example: file='DD:<USER>')
<HOME>   - The current home directory (Example: file='<HOME>/dat.bin')
```

The syntax to define a DD name is:

```
DD:name  - a DD-Statement for name (Example: file='DD:INDAT')
```

A replacement may also be used in a DD name definition:

```
Example: file='DD:<USER>'
```

A Unix path name must contain a forward slash (/). To use a file in the current directory "./filename" must be used. Paths relative to the home directory may start with a tilde character (~) as an abbreviation for <HOME> for UNIX pathnames or <SYSUID> for host dataset names.

```
Example: file='~/mydata.txt'
         file='~.po.dataset(test)'
         file='DD:~'
```

ATTENTION: Starting with version 5.1.5, the plus sign (+) can no longer be used as abbreviation for the user's home directory. The tilde (~) remains unaffected and is now the only valid abbreviation. The plus sign (+) is now a wildcard character. It was supported as alternative for EBCDIC systems, because tilde is located on different code points depending on the EBCDIC code page. Since version 5.1.5, we support the correct interpretation of diacritical characters on EBCDIC systems based on the CCSID defined in the environment variable LANG.

see *SPECIAL EBCDIC CODE PAGE SUPPORT* of Command Line Processor

To specify the casing (mainly useful for USS on ZOS) for a user ID, the following definitions can be used (if not defined as environment variable):

```
CUSER - The current user ID in upper case
Cuser - The current user ID in title case
cuser - The current user ID in lower case
```

For UNIX, LINUX, WINDOWS and other non-mainframe systems (including USS)

A filename must not be defined fully qualified, relative path are supported and the replacements above can be used. DD names are not supported and a UNIX path name is distinguished from a classic data set name in the same way as it is done in the runtime environment.

```
Example on USS:  file='//user.test(data)'
```

To use a less-than sign (<) or tilde (~) as a part of a filename, the character must be escaped by doubling it.

```
Example: file='~/my~~data.t<<t'
will expand to: file='HOME/my~data.t<t'
```

We recommend not to use special or whitespace characters in filenames.

Instead of local filenames, complete URLs defining all communication parameters and (if required) the member name can be used. The syntax is:

```
protocoll://userid:authdat@host:port/filename/?membername
FLIPN://userid:authdat&host:port/filename/:memberindex
FLIPS://userid:authdat&host:port/filename/#memberindex
FLMQS://userid:authdat@qmgr:queue/filename/?membername
```

Except the filename, all parts are optional and overwrite the already defined communication values and member names. Remote access is only available with the corresponding FLIES server.

The ampersand character (`&`) can be used instead of at sign (`@`) because the letter has different code points in EBCDIC. (the alternative character on EBCDIC, if `@` is not working, is `§` (the command line parser converts the `@` corresponding to the environment variable LANG on EBCDIC systems, if the value not set or wrong, then you can use `&` or try `§`)).

3.2.3 DIRECTORY SUPPORT

WHEN READING Multiple input files can be processed at once by using wildcard patterns as input filename and recursion into directories. It is also possible to select only certain members of FLAMFILEs by pattern or index and to specify how directories are traversed.

Input file or member names can contain the wildcards below. The wildcards are supported on subprogram level, i.e. wildcards are available in the batch utility and in subprogram calls. Wildcards are not supported on record or byte interface level.

```
'*' - zero or more characters
'+' - one or more characters
'?' - exactly one character
'%' - exactly one character (backward compatible with FLAM4 and conform with ISPF)
'#' - exactly one digit (not supported on MVS, because '#' could be part of a qualifier)
'=' - exactly one digit (supported on all platforms but mainly for MVS)
'^' - escape character (put it before a wildcard character if it is part of the file/ ↩
    member name)
```

If one of the wildcard characters is part of the filename itself, you can escape the wildcard character with the caret (`^`). For MVS datasets each qualifier is separated with a period (`.`). If you want to let the asterisk (`*`) or plus sign (+) match more than one qualifier, you must put it twice (i.e. `**` or ++). On non dataset based systems (UNIX, Windows, USS, Linux) this mechanism is not available as there are no qualifiers in filenames and hence periods are handled like any other character. Below you can find some examples:

```
file='*.txt'              - all files with extension .txt including the file named .txt
file='+.txt'              - all files with extension .txt and at least one other character
file='?.txt'              - all files with one character and the extension .txt
file='dat^*.txt'          - the file "dat*.txt"
file='**.G====V=='        - all datasets ending with GnnnnVnn (all GDG datasets)
file='<SYSUID>.**(my*)'   - all members starting with "MY" of all PO datasets under current ↩
     user
file='<SYSUID>.**(-1)'    - all previous generation of all GDG datasets under current user
```

The directory support for MVS includes Generation Data Groups (GDG). If you want to read only the previous generation of all GDG's under your SYSUID, you must type as follows:

```
file='~.**(-1)' - all GDG's with generation -1 under my SYSUID
```

The enclosing quotation marks are required because the file name string contains round brackets. Strings without quotation marks are terminated at the first space character or curved or squared bracket.

ATTENTION: If you use a wildcard in the first/high level qualifier, the catalog search interface (CSI) is used to return each dataset in the catalog. All these datasets are then compared against the corresponding pattern. This can consume a lot of time. Be careful with wildcards in the first/high level qualifier at ZOS dataset names.

ATTENTION: Using wildcards at the beginning of a pattern (for example ***(MEMBER)*) could result in opening a dataset which is stored on a tape which is not available immediately. This could result in the process waiting for console input. By default, datasets on tapes are not included in the dataset list retrieved by the wildcard mechanism. If you want to include datasets on tapes, then you must enable the corresponding switch. However, this will not prevent a console request. Files in catalog where the volume is not available, will result in the same behavior. Be careful with wildcards at the beginning of the search pattern.

The escape character can be used for datasets, but this is not required. If the match string contains parenthesis `()`, only PO datasets are matched. To match GDG, the first character inside the parenthesis must be +, − or a number. In all other cases, a PO dataset is assumed. Relative generation numbers are currently not supported for PO datasets.

To process only certain members inside archives, a match string for members can be used. A member can be defined as part of the URL with `?` as separator. Alternatively, for FLAMFILEs, a dedicated member specification can be used. In this case a leading question mark (`?`) is ignored. For a question mark as first wildcard, specify two of them.

```
file='test*.adc/?*FLAM*'    - All members that have "FLAM" in their name in files starting ↩
     with "test" and ending with ".adc"
file='*.adc member=*FLAM'   - All members that have "FLAM" at the end of their name in all ↩
     files ending with ".adc"
file='*.adc member=?FLAM*'  - All members whose names start with "FLAM" in all files  ↩
    ending with ".adc"
file='*.adc member=??FLAM*' - All members whose names start with any character followed by ↩
     "FLAM" in all files ending with ".adc"
```

You can also select members from archives based on the member index. In this case, the member name must start with a hash (`#`) or colon (`:`) and can contain a list of indexes and/or index ranges (two indices separated by minus (−), inclusive) separated by comma. For example:

```
read.flam(file=*.adc/#3,5,8-10,15)
read.flam(file=*.adc/:3,5,8-10,15)
read.flam(file=*.adc member=:3,5,8-10,15)
read.flam(file=*.adc member=#3,5,8-10,15)
```

All four variants read the 3rd, 5th, 8th, 9th, 10th and 15th member from files in the current directory that end with ".adc".

To define how wildcard strings are matched, there is an object called *DIR* that contains some switches. These include recursion into sub-directories, recursion into archives, inclusion of hidden files, symbolic link, etc. See the chapter to the *DIR* object below for more information.

WHEN WRITING Missing directories are created automatically. Overwriting of existing files can be prevented with the *REMAIN* switch (see below).

3.2.4 INPUT TO OUTPUT NAME MAPPING

When creating/extracting an archive, converting files or (more generally) doing batch processing, one might wish to automatically generate new filenames based on a pattern applied to the input filenames.

Example: You might want to read a member from a FLAMFILE which was created on a mainframe and contains dataset names. You wish to convert the dataset name into a path-based file name. You can achieve this by defining a pattern for the output file or member name.

Pattern syntax: An output filename pattern can consist of simple text strings and tokens. Text strings are used as-is as output filename. Tokens are enclosed in square brackets (`[]`) and describe processing rules applied to the original file or member name. The pattern can consist of an arbitrary number of text strings and tokens (including none at all). A token can consist of one or more processing rules. Multiple processing rules are separated by a pipe character (`|`). Every processing rule within a token receives the output of the previous rule as its input. This allows chaining of processing rules.

Characters inside tokens can be escaped by a preceding caret (^) to avoid interpretation as pattern syntax. Outside of tokens, carets are treated literally, except in front of squared brackets to treat a bracket literally and not as token delimiter.

This is the list of supported processing rules:

- [path] - Extracts the string before the last slash or backslash, i.e. the path without trailing (back)slash
- [name] - Extracts the part after the last (back)slash, i.e. the filename
- [base] - Extracts the part between the last (back)slash and the last dot, i.e. the filename without extension
- [ext] - Extracts the part after the last dot, i.e. the file extension
- [member] - Extracts a string enclosed in round brackets from the filename part of the string, e.g. a PDS member name on z/OS
- [copy] - Simply copies the input string (useful if you only want to add an extension)
- [upper] - Converts all characters to upper case
- [lower] - Converts all characters to lower case
- [title] - Converts the first character to upper case, all others to lower case
- [cutX] - Cuts off the string after the X-th character, where X is a positive or negative number indicating how many characters to keep, counting from the front (positive) or back (negative)
- [AX] - A is a place holder for a single-byte character, X is a positive or negative number; Extracts the string part between the X-th occurrence of the specified character and the next occurrence of that same character. If X is negative, search starts at the back and in reverse. If X is 0, the part before the first occurrence of A is extracted.
- [AX-] - A is a placeholder for a single-byte character, X is a positive or negative number; Extracts the string part after the X-th occurrence of the specified character. If X is negative, search starts at the back and in reverse. If X is 0, the input string is copied.
- [AX-BY] - A and B are placeholders for a single-byte character, X and Y are positive or negative numbers; Extracts the string part between the X-th occurrence of A and (relative to that occurrence) the Y-th occurrence of B. Like above, negative numbers mean counting from the back.
- [search=replace] - Replaces the first occurrence of the string *search* with the string *replace*
- [search=*replace] - Replaces all occurrences of the string *search* with the string *replace*

Examples:

```
infile='/path/to/file.ext'   pattern='text_without_tokens' outfile='text_without_tokens'
infile='/path/to/file.ext'   pattern='text^[path^]'        outfile='text[path]'
infile='/path/to/file.ext'   pattern='[path]'              outfile='/path/to'
infile='/path/to/file.ext'   pattern='[name]'              outfile='file.ext'
infile='/path/to/file.ext'   pattern='[ext]'               outfile='ext'
infile='/path/to/file.ext'   pattern='[upper]'             outfile='/PATH/TO/FILE.EXT'
infile='/path/to/file.ext'   pattern='[cut5]'              outfile='/path'
infile='/path/to/file.ext'   pattern='[/2]'                outfile='to'
infile='/path/to/file.ext'   pattern='[/-1]'               outfile='file.ext'
infile='/path/to/file.ext'   pattern='[/2-]'               outfile='to/file.ext'
infile='/path/to/file.ext'   pattern='[/3-.1]'             outfile='file'
infile='/path/to/file.ext'   pattern='[path=directory]'    outfile='/directory/to/file.ext ↩
    '
infile='USER.DATA.PDS(MYMEMBER)' pattern='[member]'        outfile='MYMEMBER'
infile='USER.DATA.PDS(MYMEMBER)' pattern='[.0|lower]/[.2|lower]/[member|lower].[ext|(0| ↩
    lower]' outfile='user/data/mymember.pds'
infile='/verylongpath/to/longfilename.ext' pattern='[path|verylong=|/=*.|.1-|upper].[base| ↩
    cut-8|upper]' outfile='PATH.TO.FILENAME'
```

More information: Patterns can be used for output file and member names. This mapping mechanism is very powerful. For example, it can be used to compress each directory into a separate FLAMFILE, with each FLAMFILE containing all files of that directory.

```
write.flam(file='[path]/?[name]')
```

ATTENTION: If you define an output filename pattern that results in the same path being generated more than once, the second and further occurrences will overwrite previously written files.

If no pattern defined, means all matching files on input, will be written to one output file or archive, the append flag is automatically enabled starting with the second file.

3.2.5 USE OF BUILT-IN FUNCTIONS

The generated description of the built-in functions are independent of the program name, owner names, command names, etc. This may make it difficult to use them. Below, you can find some examples for the use of built-in functions of FLCL.

```
To get the syntax for command CONV:
 :> flcl syntax conv

To get the syntax for the overlay read of command CONV:
 :> flcl syntax conv.read

To get the syntax for the object text of overlay read of command CONV:
 :> flcl syntax conv.read.text

To get the complete syntax (not only one level) of overlay write of
command CONV:
 :> flcl syntax conv.write all

To get the help for command INFO:
 :> flcl help info

To get the complete help for the overlay GET of command INFO:
 :> flcl help info.get all

To get the manual page for the overlay GET of command INFO:
 :> flcl help info.get man

To generate the manpage for command CONV:
 :> flcl manpage conv=manconv.txt

To generate the manual for command CONV:
 :> flcl gendocu conv=docconv.txt

To generate the complete user manual for FLCL:
 :> flcl gendocu=docflcl.txt

To change the properties for command FLAM:
 :> flcl chgprop flam mode=VR8 inrecformant=undefined cut=ON

To generate and activate a property file for command CONV:
 :> flcl genprop conv=conv.prop
 :> flcl setprop conv=conv.prop

To list all defined properties:
 :> flcl getprop

To list all properties for command INFO:
 :> flcl getprop info depall
```

```
To list only the defined properties for command INFO:
 :> flcl getprop info defall

Delete a property file for command CONV from the configuration:
 :> flcl delprop conv

To define "HUGO" as owner:
 :> flcl setowner HUGO

To get the current owner:
 :> flcl getowner

To define a environment variables:
 :> flcl setenv LANG=DE_DE.CP1252

To list all defined environment variables:
 :> flcl getenv

To delete a environment variable:
 :> flcl delenv LANG

To activate tracing in a file:
 :> flcl trace file=trace.txt
 :> flcl trace on

To deactivate the trace:
 :> flcl trace off

To list all the defined data in the configuration file:
 :> flcl config

To delete all the configuration settings from the configuration file:
 :> flcl config clear
```

All built-in functions not listed here do not require any parameters. On mainframe systems, using built-in functions in JCL looks like:

```
 //* SET ENVIROMENT VARIABLE LANG (IMPORTANT FOR AUTO CONVERSIONS)
 //SETENVL EXEC PGM=FLCL,PARM='setenv LANG=DE_DE.IBM-1141'
 //STEPLIB  DD DSN=&SYSUID..FLAM.LOAD,DISP=SHR
 //SYSPRINT DD SYSOUT=*
 //SYSOUT   DD SYSOUT=*
 //*
```

For documentation, help and syntax you can also use the ISPF commands FLDOC, FLHLP and FLSYN under menu 6.

```
 FLDOC CONV
 FLHLP XCNV.INP
 FLSYN UTIL
```

3.2.6 CONV versus XCNV and ICNV

Sometimes it is difficult to figure out which command is the right one. Especially for the conversion commands (CONV/XCNV/ICNV) understanding the differences is important:

CONV:

- simpler to use, more automatisms, fewer parameters
- the user must not know how the data is formatted or where it came from

- the conversion is done on a best-fit basis for platform specific formats
- there are no possibilities to do dirty things
- the automated detection and conversion can result in more CPU usage

XCNV:

- gives access to all capabilities of FLUC (more complex to use)
- the user must know how the inout data is formatted and where it came from
- the conversion is done as good as possible to the original data
- dirty things (e.g. manipulation of file modification time) are possible
- conversions are limited to a minimum of CPU usage

ICNV:

- provides only character set conversion for block or record-oriented files
- is much simpler than CONV and XCNV

3.3 SYNTAX

Below you can see the main syntax for this program. This content is printed on the screen if the built-in function *SYNTAX* is used.

Each command can be entered as an argument list on the command line or in a parameter file. To define a parameter file the assignment character (=) must be used. For an argument list a blank or if the command (main table) is of type overlay a dot (`.`) or if the command is of type object a round bracket open (`(`) must be given. If the round bracket open is used then the corresponding round bracket close (`)`) at the end of the command is mandatory. The dot and parenthesis are in accordance with the normal syntax.

If an argument list is used all parameters (argv[]) are concatenated to one long string and this string is passed to the parser, to allow usage of all features of the shell to build the command line input. To make sure that anything you enter is passed one by one to the parser please use double (WINDOWS®, UNIX) or single (z/OS®) quotation marks at the beginning and end of the argument list. If the dot for overlays or parenthesis for objects are used then the complete command must be included in double quotation marks. The examples below show all possible command entries for the type object:

```
:> FLCL command temp() dummy='str'
:> FLCL command(temp() dummy='str')
:> FLCL command "temp() dummy='str'"
:> FLCL "command(temp() dummy='str')"
```

For a command of type overlay it looks like:

```
:> FLCL command temp()
:> FLCL command.temp()
:> FLCL command "temp()"
:> FLCL "command.temp()"
```

In a parameter file you can start with a dot for an overlay or with parenthesis for an object or with the argument (overlay) or argument list (object) directly.

```
Syntax for program 'flcl':
--| Commands: INFO/CONV/XCNV/ICNV/XCHK/HASH/DIFF/UTIL
--|--| flcl [OWNER=oid] command "... argument list ..." [MAXCC=num]
--|--| flcl [OWNER=oid] command=" parameter file name " [MAXCC=num]
--| Built-in functions:
--|--| flcl SYNTAX [command[.path] [DEPTH1 | ... | DEPTH9 | ALL]]
--|--| flcl HELP [command[.path] [DEPTH1 | ... | DEPTH9 | ALL]] [MAN]
--|--| flcl MANPAGE [function | command[.path][=filename]] | [filename]
--|--| flcl GENDOCU [command[.path]=]filename [NONBR]
--|--| flcl GENPROP [command=]filename
--|--| flcl SETPROP [command=]filename
--|--| flcl CHGPROP command [path[=value]]*
--|--| flcl DELPROP [command]
--|--| flcl GETPROP [command[.path] [DEPTH1 | ... | DEPTH9 | DEPALL | DEFALL]]
--|--| flcl SETOWNER name
--|--| flcl GETOWNER
--|--| flcl SETENV variable=value
--|--| flcl GETENV
--|--| flcl DELENV variable
--|--| flcl TRACE ON | OFF | FILE=filename
--|--| flcl CONFIG [CLEAR]
--|--| flcl GRAMMAR
--|--| flcl LEXEM
--|--| flcl LICENSE
--|--| flcl VERSION
--|--| flcl ABOUT
--|--| flcl ERRORS
```

3.4 HELP

Here you can see the static main help for this program. This content is printed on the screen if the built-in function *HELP* is used without any further arguments.

```
Help for program 'flcl':
--| Commands - to execute powerful subprograms
--|--| flcl INFO     - Provides various information
--|--| flcl CONV     - Simplified data conversion
--|--| flcl XCNV     - Extended data conversion
--|--| flcl ICNV     - Only character conversion
--|--| flcl XCHK     - Extended data validation
--|--| flcl HASH     - Hash calculation/verification
--|--| flcl DIFF     - Compares two data sources
--|--| flcl UTIL     - Executes simply functions
--| Built-in functions - to give interactive support for the commands above
--|--| flcl SYNTAX   - Provides the syntax for each command
--|--| flcl HELP     - Provides quick help for arguments
--|--| flcl MANPAGE  - Provides manual pages (detailed help)
--|--| flcl GENDOCU  - Generates auxiliary documentation
--|--| flcl GENPROP  - Generates a property file
--|--| flcl SETPROP  - Activate a property file
--|--| flcl CHGPROP  - Change a property value in the currently active property file
--|--| flcl DELPROP  - Remove a property file from configuration
--|--| flcl GETPROP  - Show current properties
--|--| flcl SETOWNER - Defines the current owner
--|--| flcl GETOWNER - Show current owner setting
--|--| flcl SETENV   - Set an environment variable
--|--| flcl GETENV   - Show the environment variables
--|--| flcl DELENV   - Delete an environment variable
--|--| flcl TRACE    - Manage trace capabilities
```

```
--|--| flcl CONFIG   - Shows or clear all the current configuration settings
--|--| flcl GRAMMAR  - Shows the grammar for commands and properties
--|--| flcl LEXEM    - Shows the regular expressions accepted in a command
--|--| flcl LICENSE  - List license information for the program
--|--| flcl VERSION  - List version information for the program
--|--| flcl ABOUT    - Show information about the program
--|--| flcl ERRORS   - Show information about return and reason codes of the program
For more information please use the built-in function 'MANPAGE'
```

Chapter 4

Available commands

Commands are used to run powerful subprograms. The command line processor compiles a table defined syntax in a corresponding preinitialized data structure. This structure will be mapped to the parameter structure of the corresponding subprogram. The subprogram can be executed with the defined arguments/parameters.

To support these variable commands several built-in functions are available and described in the next section.

Each supported command is explained in a separate section in this document. Each section contains a synopsis including the help message, the path, the type and the syntax followed by a detailed description.

If an argument of this command is an object or overlay or, if a detailed description is available for this argument, a separate section with synopsis (help, path, type, syntax) and description is written, otherwise a bullet list is printed which contains the keyword, the syntax and the help message.

For the syntax of an overlay braces { } are used to keep all possible arguments of an overlay logically together. These braces are not part of the real syntax. The braces are only written to demonstrate clearly that one of these arguments must be selected with the DOT operator (optional) for defining the overlay.

To be compatible with certain shells, the features below are implemented:

- Strings can be enclosed with single ” or double "" quotation marks
- Integrated strings (without spaces) can also defined without quotes
- Keywords can also start with "-" or "--" in front of the qualifier
- If it unique then parenthesis and the dot can be omit for object and overlays

Commands can be declared in deep hierarchical depth. In order to simplify their handling the path is a powerful instrument for managing only relevant parts of it. Because of that the path is printed for each synopsis.

To run commands under different owners, the owner id can be defined in front of the command.

For job control, the maximum condition code (MAXCC=num) of the command execution can optionally be set as the last parameter for each command.

```
flcl [OWNER=oid] command "... argument list ..." [MAXCC=num]
flcl [OWNER=oid] command=" parameter file name " [MAXCC=num]
```

The parameter for each command can be provided as argument list on the command line or as parameter file.

4.1 COMMAND *INFO*

SYNOPSIS

```
HELP:   Provides various information
PATH:   LIM.flcl
TYPE:   OBJECT
SYNTAX: :> flcl INFO(GET.{},FMT=LIST/XML,OUT='str'/STDOUT/STDERR,DIR(),LOG())
```

DESCRIPTION The INFO command provides further information about files, CCSIDs among others. To use CONV and other commands it is sometimes useful to know which CCSIDs are supported or which members are in a FLAMFILE or how the file attributes are named.

To get syntax information, please use:

```
   flcl SYNTAX INFO
```

To get help for a parameter, please use:

```
   flcl HELP INFO.parameter[.parameter[...]]
```

To read the manual page for a parameter, please use:

```
   flcl MANPAGE INFO.parameter[.parameter[...]]
      or
   flcl HELP INFO.parameter[.parameter[...]] MAN
```

To generate the user manual for a command, please use:

```
   flcl GENDOCU INFO=filename
```

Parameters can be defined by the command line (directly or per file) or by properties read from the corresponding property file.

EXAMPLES

```
   flcl INFO get.ccsids
   flcl INFO get.file='test.adc' out='list.txt'
   flcl INFO get.file='~.*' dir(rec) out='list.txt'
   flcl INFO get.fkme(help)
```

ARGUMENTS

- NUMBER:FMT=LIST/XML -Specify formatting of informations [LIST]
  - LIST -Provide information as list
  - XML -Provide information in XML-structure

4.1.1 OVERLAY *GET*

SYNOPSIS

```
HELP:   Specify what you want to get
PATH:   INFO
TYPE:   OVERLAY
SYNTAX: GET.{CCSIDS/ENCODINGS/FKME()/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION With the overlay GET you can choose which information will be printed.

ARGUMENTS

- SWITCH:CCSIDS -List of supported CCSIDs
- SWITCH:ENCODINGS -List of all encodings
- STRING:FILE='str'/STREAM/DUMMY -Information about a file (flam4/gzip/xz/bzip2/...)
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.1.2 OBJECT *FKME*

SYNOPSIS

```
HELP:   List informations, help and documentation of FKME
PATH:   INFO.GET
TYPE:   OBJECT
SYNTAX: FKME(LIBRARY='str',FUNCTION='str',HELP,DOCU)
```

DESCRIPTION Through the object FKME you can define the library and function of an FKME module to get information, help and documentation of the corresponding FLAM Key Management Extension. By default, the *libfkme* of FLAM is used. If no function name is specified, all known entries of this library are used. This works only for *libfkme* as we do not know the function names for other libraries.

The function codes for help and documentation might not be implemented by other FKMEs, i.e. these switches are mainly useful with the *libfkme* from limes datentechnik.

To get FKME info for load modules on mainframes, which must be fetched from STEPLIB, you must type a * for the library name.

EXAMPLES

```
 info get.fkme                            # list all info messages
 info get.fkme(lib=*)                     # list info of FKMESTD0 load module
 info get.fkme(lib=* func=FKMESWE0)       # list info of FKMESWE0 load module
 info get.fkme(func=SYMNP11 help)         # list help for function SYMNP11 of libfkme
 info get.fkme(docu)                      # list docu all functions in libfkme
```

ARGUMENTS

- STRING:LIBRARY='str' -Library name [libfkme]
- STRING:FUNCTION='str' -Function name (entry)
- SWITCH:HELP -Get help message
- SWITCH:DOCU -Get documentation

4.1.3 PARAMETER *OUT*

SYNOPSIS

```
HELP:   Name of the output file [LOG]
PATH:   INFO
TYPE:   STRING
SYNTAX: OUT='str'/STDOUT/STDERR
```

DESCRIPTION With the OUT parameter you can choose the file (as STRING) or stream (STDOUT/STDERR) where the information is printed.

SELECTIONS

- STDOUT -Write output to stdout
- STDERR -Write output to stderr

4.1.4 OBJECT *DIR*

SYNOPSIS

```
HELP:   Defines how to handle directories and various file types if wildcards used
PATH:   INFO
TYPE:   OBJECT
SYNTAX: DIR(LINK,ALIAS,HIDDEN,RECURSIVE,ARCHIVE,ONEFILESYSTEM,TAPE)
```

DESCRIPTION This object defines multiple switches that control how input files are found. This is especially important when using wildcards in input filenames.

There are switches to enable recursion into subdirectories, following symbolic links, finding hidden files, recursion into archive files, include files on tapes, resolving aliases and enforcing to not leave the filesystem.

If the recursive flag is set, the filename portion of the input file string is searched for recursively. If a directory path containing wildcards is present, the filename portion is searched for recursively in all paths that match the directory path pattern, e.g. "**/path/**.txt" matches my/path/hugo.txt, another/path/hugo.txt as well as my/path/subdir/hugo.txt. Without the recursive flag, the last file would not be found.

If the archive switch is enabled, FLAMFILES, TARBALLS, ZIP files and other kinds of supported archives are searched. All the matching members are extracted. In other words, archives are interpreted like subdirectories if the flag is enabled.

If the tape switch is enabled, then datasets stored on tapes are searched. If the name of such a file matches the pattern, then it might occur that console input and a manual intervention is required to retrieve the respective dataset(s).

If the link switch is enabled, symbolic links are followed (otherwise they are ignored). Links are mainly supported on modern file systems. For dataset aliases on mainframe systems, see the alias flag.

With the alias switch enabled, datasets names which are aliases of the real dataset are resolved to their actual names (otherwise they are ignored). Dataset aliases only exist on mainframe systems. This flag has no effect on other platforms.

If a transparent read method (e.g. read.auto()) is used, then the archive flag is enabled by default and cannot be disabled.

ARGUMENTS

- `SWITCH:LINK -Include symbolic links`
- `SWITCH:ALIAS -Resolve data set aliases`
- `SWITCH:HIDDEN -Include hidden files`
- `SWITCH:RECURSIVE -Include sub directories`
- `SWITCH:ARCHIVE -Include archives (FLAMFILEs, ZIP, ...)`
- `SWITCH:ONEFILESYSTEM -Don't cross file system boundaries`
- `SWITCH:TAPE -Include files deposited on tapes`

4.1.5 OBJECT *LOG*

SYNOPSIS

```
HELP:   Logging Parameter
PATH:   INFO
TYPE:   OBJECT
SYNTAX: LOG(MESSAGE(),DESTINATION.{})
```

DESCRIPTION The LOG parameter allows flexible adjustment for the logging of messages. It is possible to specify the destination as well as which kind of message will be written.

For more information please have a look at the man pages for each parameter of LOG.

4.1.6 OBJECT *MESSAGE*

SYNOPSIS

```
HELP:   Select type of messages to log
PATH:   INFO.LOG
TYPE:   OBJECT
SYNTAX: MESSAGE(NONE,ERROR,ERRTRACE,WARNING,NOTICE,DEBUG,ABOUT,VERSION,INPUT,PARAMETER, ↩
    STATISTIC,INFO,LICID,LICENSE,PROGRESS,ALL,DEFAULT,MINIMAL,ERRONLY)
```

DESCRIPTION The log message specification is a classification of the messages to write to the selected log destination. Each type is selectable individually by adding its argument inside this object.

For added convenience, the special types NON, ALL, MINIMAL select the proper message types with just one argument.

If no message specification is given, a DEFAULT is used which writes out messages of these types:

- ERROR
- ERRTRACE
- WARNING
- NOTICE
- PARAMETER
- STATISTIC
- INFO
- LICID

ARGUMENTS

- `SWITCH:NONE -No messages will be logged`
- `SWITCH:ERROR -Error messages will be logged`
- `SWITCH:ERRTRACE -Error trace will be logged`
- `SWITCH:WARNING -Warnings will be logged`
- `SWITCH:NOTICE -Notices will be logged`
- `SWITCH:DEBUG -Debug information will be logged`
- `SWITCH:ABOUT -About information will be logged`
- `SWITCH:VERSION -Version information will be logged`
- `SWITCH:INPUT -Input data will be logged (dangerous for passwords)`
- `SWITCH:PARAMETER -Parsed parameters will be logged`
- `SWITCH:STATISTIC -Statistics will be logged`
- `SWITCH:INFO -Requested information will be logged`
- `SWITCH:LICID -License ID will be logged`
- `SWITCH:LICENSE -License summary will be logged`
- `SWITCH:PROGRESS -Progress bar will be logged`
- `SWITCH:ALL -All messages are logged`
- `SWITCH:DEFAULT -Relevant messages are logged`
- `SWITCH:MINIMAL -Only important messages are logged`
- `SWITCH:ERRONLY -Only errors and error trace are logged`

4.1.7 OVERLAY *DESTINATION*

SYNOPSIS

```
HELP:   Destination for log messages
PATH:   INFO.LOG
TYPE:   OVERLAY
SYNTAX: DESTINATION.{STREAM()/FILE()/SYSTEM()}
```

DESCRIPTION The log destination parameter defines where the log messages will be written.

Each log message starts with the current time and date.

All the possible destinations allow to specify a custom identification string which will be printed after time and date of each message. This is done with the IDENT argument.

If no IDENT specified the default will be *owner.program.command*

If no log destination defined stream will be used as default.

4.1.8 OBJECT *STREAM*

SYNOPSIS

```
HELP:   Log to a stream
PATH:   INFO.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: STREAM(FORMAT=STANDARD/DIALOG,IDENT='str',STDOUT)
```

DESCRIPTION The STREAM object allows the log messages to be written to the standard output or the standard error output. Standard error output is used if STDOUT is not specified.

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- SWITCH:STDOUT -Log messages to STDOUT instead of STDERR

4.1.9 OBJECT *FILE*

SYNOPSIS

```
HELP:   Log to a flat file
PATH:   INFO.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: FILE(FORMAT=STANDARD/DIALOG,IDENT='str',NAME='str')
```

DESCRIPTION The flat file option allows the log messages to be written to a normal file. The name of the file must be specified for this destination. The log messages will be appended to this file if it already exists.

For the log file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   NAME='~.MYLOG(LOG1)'                  ; PDS on z/OS
   NAME='<SYSUID>.MYLOG02'               ; PS dataset on z/OS
   NAME='~/mylog3.txt'                   ; Unix path name
   NAME='<HOME>\logs\mylog4.txt'         ; Windows path name
```

ATTENTION: Parallel process cannot share the same log file

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- STRING:NAME='str' -Filename for appending log messages

4.1.10 OBJECT *SYSTEM*

SYNOPSIS

```
HELP:   Use system logging facility
PATH:   INFO.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: SYSTEM(IDENT='str')
```

DESCRIPTION The SYSTEM destination allows the log messages to be written to the logging system of the operating system.

On Unix and z/OS® systems the syslog() call is used. On Windows® the event logging facility is used.

```
NOTE: On Windows(R) this requires the registry key entry:
      HKEY_LOCAL_MACHINE
        SYSTEM
          CurrentControlSet
            Services
              EventLog
                Application
                  FLM
```

This registry key will be created by using the DLL flevent.dll

ARGUMENTS

- STRING:IDENT='str' -Identifier used to identify log entries

4.2 COMMAND *CONV*

SYNOPSIS

```
HELP:   Simplified data conversion
PATH:   LIM.flcl
TYPE:   OBJECT
SYNTAX: :> flcl CONV(READ.{},WRITE.{},DIR(),LOGGING.{},MESSAGE(),INVERSE='str'/STDOUT/ ↩
    STDERR)
```

DESCRIPTION The CONV command converts data between various formats with an easy to use syntax. It was designed to simplify the use of the powerful Frankenstein Limes Universal Converter (FLUC). It can be used to read local or remote data sources (requires a FLIES server), convert the data (e.g. decrypt, decompress and character conversion) and write the result to a local or remote sink.

As an example, the CONV command can be used to read an encoded, encrypted and compressed text file in UTF-16 from a Windows system, generate or verify a checksum and write each line as text record into a mainframe dataset with a dedicated record format (FB, VB, ...) in EBCDIC with ASA control characters. All these conversions are done in one step to reduce CPU and I/O consumption.

The following features are supported:

- Different kinds of remote access protocols (IP-Native, MQ-Series, SSH)
- Different kinds of data sources (Files, Streams, Tables)
- Different kinds of archive formats (ZIP, FLAMFILE, TAR)
- Different kinds of I/O methods (Binary, Character, Text, Record, FLAM)
- Different data encodings (HEX (Base16), Base32, Base64)
- Different encryption methods (FLAM, PGP)
- Different compression methods (GZIP, BZIP2, XZ)
- Automatic decoding of data encodings during read operations
- Auto-detection and decoding of length fields in data blocks
- Auto-detection of character sets and language based conversion
- Powerful character conversion based on CCSIDs with subset support and reporting
- Different logical data formatting (BIN, TEXT, XML, ...)

If the CONV command is not sufficient to meet your requirements, you can use the XCNV command which provides all full conversion capabilities of FLAM in a kind of expert mode with maximum configurability. The CONV command provides an easy to use subset of all conversion capabilities. In contrast to XCNV, the CONV command performs various types of auto detection and automated conversions to get a data representation that fits the current system and "makes sense" in the given context. To use the XCNV command, you must know what kind of data is read and can fully control how the data is written.

To get syntax information, please use:

```
flcl SYNTAX CONV
```

To get help for a parameter, please use:

```
flcl HELP CONV.parameter[.parameter[...]]
```

To read the manual page for a parameter, please use:

```
flcl MANPAGE CONV.parameter[.parameter[...]]
   or
flcl HELP CONV.parameter[.parameter[...]] MAN
```

To generate the user manual for the command, please use:

```
flcl GENDOCU CONV=filename
```

Parameters can be defined via command line (directly or by parameter file) or via properties taken from the corresponding property file

EXAMPLE

```
flcl CONV read.text(file='swift.txt.gz' hash(algo=SHA256))
          write.record(file='<SYSUID>.text(swift)'
                       chrmode=SUB systab=iconv usrtab=DELA
                       report='<SYSUID>.report(swift)'
                       format=VBA ccsid='IBM273')
```

Reads a binary file, calculates a SHA-256 checksum in GNU style, automatically decompresses the data (as text cannot be compressed), performs a character conversion to UTF-8 (no report required), formats the data into text records and rest elements, convert the text records to old German EBCDIC (IBM-273) with transliteration, substitution and a report file (upgrade-proof) and writea the text records to a variable blocked dataset with ASA print control characters. The user table *DELA* ensures that only the subset of characters are valid which are available in CP1252, IBM-1141 and ISO-8859-15.

```
flcl CONV read.record(file='<SYSUID>.text(swift)' ccsid='IBM273')
          write.text(method=CRLF, ccsid='LATIN1', comp.bzib())
```

Reads records from a VBA dataset, does a character conversion to UTF-8, formats the data as text with carriage return and line feed as delimiter, converts the text data to Latin-1 and writes the result to BZIP file. Conversion stopped if a character is not convertible. Assuming the input file is the output of the previous example, this cannot be the case because the user table DELA used to create the file *<SYSUID>.text(swift)* prevents characters not available in Latin-1 (a.k.a. ISO-8859-1). The previous example would transliterate the EURO sign (€) to *EUR* because the CCSID was IBM-273 (does not include the EURO sign). To retain the EURO sign, the CCSID IBM-1141 (=IBM273+EURO) would have to be used in the previous example and ISO-8859-15 (=LATIN1=ISO-8859-1+EURO) in this example.

```
flcl CONV read.file='my.data'
          write.record(file=stream, ccsid='IBM1141'
                       recm=cut,chrm=sub,systab=iconv)
```

Reads XML, text or binary data in encoded or clear form and converts the data to records which are printed to SYSOUT (STREAM) in EBCDIC (IBM1141) where longer records are truncated (RECMODE=CUT), not convertible characters are transliterated (SYSTAB=ICONV) and if this not possible substituted (CHRMODE=SUB). With this command you can read any kind of file on mainframes and convert it to readable records. This command is used as part of the ISPF line command FLVIEW. It reads normal record oriented host datasets (PS, PDS(E) as VB/FB/UNDEFINED with or without ASA or machine print control character), VSAM-ESDS/RRDS/KSDS, FLAMFILEs, binary transfered text or XML files from other platforms, encoded and or compressed text or XML files, aso.

```
flcl INFO get.file='my.data'
```

Lists all determined file attributes including CCSID, delimiter, possible base encodings and (if appropriate) the member list of (concatenated) FLAMFILEs, GZIP, BZIP2 or XZ files.

4.2.1 OVERLAY *READ*

SYNOPSIS

```
HELP:   Inbound parameter to read data
PATH:   CONV
TYPE:   OVERLAY
SYNTAX: READ.{BINARY()/CHARACTER()/TEXT()/XML()/RECORD()/FLAM4()/AUTO()/FILE='str'/STREAM/ ↩
    DUMMY}
```

DESCRIPTION With the overlay READ you can choose the kind of data (binary, text, character, record, FLAM4FILE, table, ...) to read or you can use *auto* for auto detection of the file/data format. For each read method, a dedicated behavior is implemented and the corresponding parameters can be used to detail the procedure. For example: If we know it is text and the data is compressed then the data will be automatically de-compressed, because compressed data cannot be converted to text records. If the compressed data is, for example, Base64 encoded, then a decoding is done before decompression and so on. Based on the knowledge about the kind of data you want to read, we have simplified the way to automatically do the right things for you. Reading files transparently for each kind of data format can be very powerful.

After the read operation, you can choose the same or another write operation to convert your data between different platforms, file and data formats, character sets, compression, encryption or encoding methods.

Additionally you can calculate or verify checksums on the binary data stream direct after the read operation.

Below, you can find the currently supported data sources explained in a dedicated chapter.

4.2.2 OBJECT *BINARY*

SYNOPSIS

```
HELP:   Read binary data from a file
PATH:   CONV.READ
TYPE:   OBJECT
SYNTAX: BINARY(FILE='str'/STREAM/DUMMY,BLKSIZE=num,DECODE[=NONE/FLAMFILE/FIODEC/DECRYP/ ↩
    CRYDEC/DECOMP/CMPDEC/ALWAYS],DECRYPT[{}...],FDECODE/COD.{},RLDECO,PRNCONTROL=DETACH/ ↩
    RETAIN/ERASE,SUBSYSTEM(),FRCBLK,REMOVE,PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK ↩
    )
```

DESCRIPTION Read binary operates on blocks of binary data. By default, no decompression or any other conversions are performed. These must explicitly be turned on using the respective parameters.

If the *decode* parameter is set, the data is automatically decrypted and uncompressed. If the encrypted or compressed data is encoded (e.g. Base64), it is decoded before uncompression/decryption. If a 4 byte record length format detected and decoding should be done, then the records are parsed before the next decoding step. If the uncompressed/decrypted data is encoded, it is only decoded if *decode* is set to *CMPDEC* or *ALWAYS*. By setting *decode* to one of the available values, you have further control over the enabled automatisms.

Binary data blocks are the simplest form of data type handled by FLAM. Binary data can only be accessed by byte offset. The structure of the data is unknown to FLAM.

Using only binary blocks with FLAM mainly equals the behavior and abilities of other encryption and compression tools. You may want to try one of the other available read methods to read structured data.

ARGUMENTS

- STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- NUMBER:DECODE=NONE/FLAMFILE/FIODEC/DECRYP/CRYDEC/DECOMP/CMPDEC/ALWAYS -Decode encoded, encrypted or compressed data [0=NONE]
  - NONE -No automated decoding
  - FLAMFILE -Transparent read of FLAMFILEs
  - FIODEC -Decoding after file IO before decryption
  - DECRYP -Decryption of encrypted data
  - CRYDEC -Decoding after decryption before decompression
  - DECOMP -Decompression of compressed data
  - CMPDEC -Decoding after decompression in front of formatting
  - ALWAYS -Always decode encoded data (causes the same as DECOMP)
- SWITCH:RLDECO -Activate an additional record length decoding [FALSE]
- SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.2.3 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size to read [AUTO]
PATH:   CONV.READ.BINARY
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.2.4 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   CONV.READ.BINARY
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.2.5 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   CONV.READ.BINARY.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['******']`
  - `DEFAULT -FLAM4 default password`

4.2.6 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   CONV.READ.BINARY.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME [”]`

4.2.7 OVERLAY *FDECODE*

SYNOPSIS

```
HELP:   Additional enforced decoding [NONE]
PATH:   CONV.READ.BINARY
TYPE:   OVERLAY
SYNTAX: FDECODE/COD.{BASE64/OPGP/BASE32/BASE16}
```

DESCRIPTION The overlay *fdecode* allows you to add an additional decoding step after auto decoding (*decode* parameter defined) was done.

On *read.bin()* with the *decode* parameter set, encrypted data (PGP, ...) is automatically decrypted and compressed data (GZIP, ...) is decompressed if the method is supported by FLAM. If the encrypted or compressed data is base encoded, it will decoded before decompression or decryption. However, there is no automatic base decoding of the decompressed or decrypted data since we are in binary mode and, therefore, it is impossible to guess whether you want to read base encoded or decoded data. The data might not even be base encoded, but just look like it is.

fdecode allows you to enforce base decoding after auto decoding by specifying the base to use. If you specify a base, but the data is not encoded with this base, you will get an error.

If you define *decode=CMPDEC* or *decode=ALWAYS* then a base decoding after decompression is done, if a base decoding is possible. This option can be used on behalf of *fdecode*, but auto detection of the correct base may fail. Therefore, it is suitable to use the *fdecode* parameter if you already know the base.

The enforcement of base decoding can even be used if no auto decoding is activated (i.e. *decode* parameter not specified).

See below for the decoding possibilities.

ARGUMENTS

- `SWITCH:BASE64/OPGP -Base64 decoding (RFC4648)`
- `SWITCH:BASE32 -Base32 decoding (RFC4648)`
- `SWITCH:BASE16 -Base16 decoding (RFC4648)`

4.2.8 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   CONV.READ.BINARY
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- DETACH –Manage print control character as record attribute
- RETAIN –Leave print control character as part of the data record
- ERASE –Erase the print control character from the record

4.2.9 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   CONV.READ.BINARY
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- STRING:NAME='str' –Name of the subsystem
- STRING:PARM='str' –Parameter list for the subsystem (comma separated)

4.2.10 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   CONV.READ.BINARY
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- WIN -Choose Windows as platform (CRLF)
- UNX -Choose Unix derivative as platform (LF)
- ZOS -Choose ZOS as platform (NL)
- USS -Choose USS of ZOS as platform (NL)
- VSE -Choose VSE as platform (NL)
- BS2 -Choose BS2000-OSD as platform (NL)
- MAC -Choose MAC as platform (CR)

4.2.11 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   CONV.READ.BINARY
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)

  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.2.12 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   CONV.READ.BINARY.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.2.13 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   CONV.READ.BINARY
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.2.14 OBJECT *CHARACTER*

SYNOPSIS

```
HELP:   Read character data from a file
PATH:   CONV.READ
TYPE:   OBJECT
SYNTAX: CHARACTER(FILE='str'/STREAM/DUMMY,BLKSIZE=num,CCSID='str'/DEFAULT/ASCII/EBCDIC/ ↩
    BOMUTF/BOMUCS,BOM,ENL2LF,DECODE=NONE/FIODEC/CRYDEC/CMPDEC/ALWAYS,DECRYPT[{}...], ↩
    PRNCONTROL=DETACH/RETAIN/ERASE,SUBSYSTEM(),FRCBLK,REMOVE,LANG='str',PLATFORM=WIN/UNX/ZOS ↩
    /USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read char operates on blocks of text data. If the input data is encoded (e.g. Base64), encrypted or compressed, it is automatically decoded, decrypted and/or decompressed. Through the *decode* parameter, you can limit the number of encoding layers to remove (for example to retrieve a Base64-encoded text in a GZIP file instead of the decoded text). If a 4 byte record length format is detected, the record lengths are removed to build a valid character block.

For decryption, you must provide the required decryption parameters, e.g. a key reference or passphrase. Several decryption methods can be enabled at once with the corresponding parameters and will be used to transparently read the data.

The character data is converted from the provided CCSID to UTF-8. If no CCSID is provided or auto detection fails, the default CCSID is determined in a system-dependent way in the following order:

- Stored CCSID from the file system (if available)
- From the environment variable LANG
- ISO-8859-1 (LATIN1) on ASCII and IBM-1148 (intl) on EBCDIC systems

The character data blocks can be converted to any code page on write. If you don't need character conversion, the binary version of reading files might be more suitable.

In contrast to the read text method, the read char method does not scan for line delimiters to build valid records. This means that the data will remain a chunk of character data.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`
- `NUMBER:DECODE=NONE/FIODEC/CRYDEC/CMPDEC/ALWAYS -Decode encoded data first (remove ba se encodings)`

  - NONE -No automated decoding
  - FIODEC -Decoding after file IO before decryption
  - CRYDEC -Decoding after decryption before decompression
  - CMPDEC -Decoding after decompression in front of formatting
  - ALWAYS -Always decode encoded data (causes the same as DECOMP)
- SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.2.15 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size for read operations [AUTO]
PATH:   CONV.READ.CHARACTER
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.2.16 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID [auto detection]
PATH:   CONV.READ.CHARACTER
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)

- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (auto-detect)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`
- `BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)`
- `BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)`

4.2.17 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   CONV.READ.CHARACTER
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.2.18 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   CONV.READ.CHARACTER
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.2.19 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   CONV.READ.CHARACTER
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.2.20 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   CONV.READ.CHARACTER.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
 * In hexadecimal format (recommended): password=x'hex'
 * In ASCII format: password=a'ascii'
 * In EBCDIC format: password=e'ebcdic'
 * In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']`
  - `DEFAULT -FLAM4 default password`

4.2.21 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   CONV.READ.CHARACTER.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- STRING:LIBRARY='str' -Library name for FKME ['libfkme']
- STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']
- STRING:PARAMETER='str' -Parameter for the call of FKME [”]

4.2.22 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   CONV.READ.CHARACTER
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- DETACH -Manage print control character as record attribute
- RETAIN -Leave print control character as part of the data record
- ERASE -Erase the print control character from the record

4.2.23 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   CONV.READ.CHARACTER
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.2.24 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   CONV.READ.CHARACTER
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.2.25 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   CONV.READ.CHARACTER
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- WIN -Choose Windows as platform (CRLF)
- UNX -Choose Unix derivative as platform (LF)
- ZOS -Choose ZOS as platform (NL)
- USS -Choose USS of ZOS as platform (NL)
- VSE -Choose VSE as platform (NL)
- BS2 -Choose BS2000-OSD as platform (NL)
- MAC -Choose MAC as platform (CR)

4.2.26 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   CONV.READ.CHARACTER
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)

  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.2.27 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   CONV.READ.CHARACTER.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.2.28 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   CONV.READ.CHARACTER
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.2.29 OBJECT *TEXT*

SYNOPSIS

```
HELP:   Read text data from a file
PATH:   CONV.READ
TYPE:   OBJECT
SYNTAX: TEXT(FILE='str'/STREAM/DUMMY,BLKSIZE=num,RECLENGTH=num,SUPTWS,NELDLM,CCSID='str'/ ↩
    DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,BOM,ENL2LF,RPLFFD=num,PADCHAR=num,DECODE=NONE/FIODEC/ ↩
    CRYDEC/CMPDEC/ALWAYS,DECRYPT[{}...],PRNCONTROL=DETACH/RETAIN/ERASE,SUBSYSTEM(),FRCBLK, ↩
    REMOVE,BINERROR,CHRERROR,LANG='str',PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read text works on blocks of text data or records. First, the text data is converted to the UTF-8 character set. Then the text data is split into record and rest elements based on text delimiters or record length. The data must contain a valid text delimiter within the provided record/ line length. If the data contains 4 byte length fields, the length fields are used to build a block as list of records. If no delimiter is found, the record length are used to form the text records. If the data is block-oriented (no record length) and neither delimiters nor 4 byte length fields are found, the data will be wrapped into records of the provided record length.

A text record contains all text between two delimiters, but without trailing whitespace if SUPTWS used. Trailing whitespace, the delimiter and possibly some padding characters make up the rest element. Taken together, the record and the rest elements represent the original data.

List of valid text delimiters (UNICODE-Points):

- 0x00 (padding character (can be changed))
- 0x0A (carriage return (EBCDIC 0x25))
- 0x0D (line feed (EBCDIC(0x0D)))
- 0x0A, 0x0D (carriage return line feed)
- 0x0A, . . . , 0x0A, 0x0D (dirty delimiters)
- 0x0A, 0x0D, . . . , 0x0D (dirty delimiters)
- 0x0C (form feed (EBCDIC 0x22))
- 0x85 (new/next line (EBCDIC 0x15, optional if single byte ASCII))

0x85 are used for UTF-8 (C285) and EBCDIC (0x15) but in single byte ASCII code pages the NELDLM flag must be defined to scan for 0x85, because 0x85 normally used as currency sign accept for ISO code pages. To prevent EBCDIC(0x15) to UNICODE(0x85) at character conversion FLAM supports the ENL2LF and ELF2NL switches.

If the text data is detected as being compressed, it is decompressed automatically. For encodings like Base64, automated decoding is useful in most cases. However, if you want to read the base encoded data as text, you must set the *decode* parameter to the number of automatic decodings to perform. For a transparent decryption you, must provide the required parameters. Decryption requires at least a key reference as parameter. Several decryption methods can be enabled at once with the corresponding parameters. 4 byte length fields in the data are detected and cannot be part of valid text data. Therefore, these length fields are used automatically to build a list of records.

The character data is converted from the provided CCSID to UTF-8. If no CCSID is provided, auto detection is applied. If this not successful, a system dependent default CCSID is used: If available, the CCSID stored in the file system is used. If this is unset or not supported, the environment variable LANG is used. If this is also not set to a valid value, ISO-8859-1 (Latin-1) is used on ASCII and IBM-1148 (international) on EBCDIC systems.

While text formatting, form feed characters may optionally be replaced by empty records.

A padding character may be defined which is regarded as a special delimiter. Consecutive occurrences of this padding character are regarded as one delimiter, i.e. the text "some text_more text" will become two records "some text" and "more text". Not being a part of the original data, padding characters are simply dropped and are not part of the rest element.

With read.text() you can transparent read normal text files with delimiter in clear, compressed (GZIP, BZIP2, XZ(LZMA)), encrypted (FLAM, PGP) or encoded (Base64/32/16) form, normal record oriented data sets (FB(A/M)/VB(A/M)/VSAM/...) or FLAMFILEs.

ARGUMENTS

- STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- NUMBER:RECLENGTH=num -Maximum record length (Host) /line length (Unix/Win) for text parsing [512]
- SWITCH:SUPTWS -Suppress trailing whitespaces [FALSE]
- SWITCH:NELDLM -Activate NEL (0x85) as delimiter for ASCII character sets [FALSE]
- NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assuming n lines per page [0 =no replacement]
- NUMBER:PADCHAR=num -Padding character to be regarded as additional delimiter [0x00]
- NUMBER:DECODE=NONE/FIODEC/CRYDEC/CMPDEC/ALWAYS -Decode encoded data first (remove base encodings)
  - NONE -No automated decoding
  - FIODEC -Decoding after file IO before decryption
  - CRYDEC -Decoding after decryption before decompression
  - CMPDEC -Decoding after decompression in front of formatting
  - ALWAYS -Always decode encoded data (causes the same as DECOMP)
- SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]
- SWITCH:BINERROR -Enforce an error if binary data detected [FALSE]
- SWITCH:CHRERROR -Enforce an error if character data without delimiter detected [FALSE]

4.2.30 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size for read operations [AUTO]
PATH:   CONV.READ.TEXT
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.2.31 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID [auto detection]
PATH:   CONV.READ.TEXT
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- DEFAULT -Use default CCSID (auto-detect)
- ASCII -Use default ASCII CCSID (environment)
- EBCDIC -Use default EBCDIC CCSID (environment)
- BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)
- BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)

4.2.32 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   CONV.READ.TEXT
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.2.33 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   CONV.READ.TEXT
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.2.34 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   CONV.READ.TEXT
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
 decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
 decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.2.35 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   CONV.READ.TEXT.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['******']`
  - `DEFAULT -FLAM4 default password`

4.2.36 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   CONV.READ.TEXT.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME [”]`

4.2.37 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   CONV.READ.TEXT
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.2.38 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   CONV.READ.TEXT
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.2.39 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   CONV.READ.TEXT
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.2.40 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   CONV.READ.TEXT
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.2.41 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   CONV.READ.TEXT
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.2.42 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   CONV.READ.TEXT.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- `STRING:VALUE='str' -Checksum as hex-string`
- `STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`

4.2.43 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   CONV.READ.TEXT
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.2.44 OBJECT *XML*

SYNOPSIS

```
HELP:   Read XML data from a file
PATH:   CONV.READ
TYPE:   OBJECT
SYNTAX: XML(FILE='str'/STREAM/DUMMY,BLKSIZE=num,NOCMNT,CCSID='str'/DEFAULT/ASCII/EBCDIC/ ↩
    BOMUTF/BOMUCS,BOM,ENL2LF,RPLFFD=num,DECRYPT[{}...],SUBSYSTEM(),FRCBLK,REMOVE,LANG='str', ↩
    PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read XML works on blocks of binary or text data. Character conversion takes place before processing the XML data. If no CCSID is provided, then auto detection is used. If UTF-8 is detected, character conversion is skipped. Line delimiters must be one of 0x0A, 0x0D or 0x0D0A (after conversion to UTF-8). If a CCSID is supplied, the character data is converted from the provided CCSID to UTF-8 before performing XML processing, using the supplied CCSID as source encoding. If the input data is encoded (e.g. Base64), encrypted, compressed or contains 4 byte length fields, it is automatically decoded, decrypted, and/or decompressed to build a valid XML data block. During parsing, all line delimiters are normalized to line feed characters (0xA) as defined by the XML specification.

The XML data is parsed using the Expat library and transformed into FLAM elements, which allows various formatting options when writing, including minimizing, pretty printing XML data and restoring an equivalent of the original document.

On some EBCDIC machines, if character conversion from an EBCDIC charset is used, the new line character (0x15) is not properly converted, causing the XML parser to fail. In this case, turn on the enl2lf to enable proper conversion of new lines (0x15) to line feeds (0xA).

If reading XML, the semantics of write modes (write.*()) change as follows:

- BINARY: The FLAM elements produced by this read mode are used to restore the original XML file as close as possible (standard formatting). The XML will be in UTF-8 and lines are delimited by line feed (0xA).
- CHARACTER: The FLAM elements are used to produce a minimized version of the XML data. Line breaks and whitespace are removed unless they are part of actual character data.
- TEXT: Human readable XML data with proper indentation is produced (pretty printing). Linebreaks are converted to a system-specific representation.
- XML: Produces XML data using the specified formatting method. The XML data will be in UTF-8 and lines are delimited by line feed (0xA) unless explicit character conversion is enabled.
- RECORD: Human readable XML data with proper indentation is produced (pretty printing). The produced XML data is separated into records, each record containing one line of XML.
- FLAM4: Same as RECORD, but stored in a FLAM4 file.

Known limitations:

- If the attributes inside an XML tag are separated by more than one space character, the additional space characters are dropped (as per XML specification).
- Due to a parser limitation, declared entities that appear inside attributes will be output in its substituted form.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`
- `SWITCH:NOCMNT -Ignore XML comments [FALSE]`
- `NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assumin g n lines per page [0 =no replacement]`
- `SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]`
- `SWITCH:REMOVE -Remove original file after successful processing [FALSE]`

4.2.45 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size for read operations [AUTO]
PATH:   CONV.READ.XML
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.2.46 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID [no character conversion, UTF-8/US-ASCII expected]
PATH:   CONV.READ.XML
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- DEFAULT -Use default CCSID (auto-detect)
- ASCII -Use default ASCII CCSID (environment)
- EBCDIC -Use default EBCDIC CCSID (environment)
- BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)
- BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)

4.2.47 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   CONV.READ.XML
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.2.48 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   CONV.READ.XML
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.2.49 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   CONV.READ.XML
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
 decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
 decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.2.50 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   CONV.READ.XML.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['******']`
  - `DEFAULT -FLAM4 default password`

4.2.51 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   CONV.READ.XML.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME [”]`

4.2.52 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   CONV.READ.XML
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.2.53 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   CONV.READ.XML
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.2.54 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   CONV.READ.XML
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.2.55 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check  of one way hash values of input data
PATH:   CONV.READ.XML
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.2.56 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   CONV.READ.XML.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                            ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'                ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'               ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                  ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'       ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.2.57 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   CONV.READ.XML
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.2.58 OBJECT *RECORD*

SYNOPSIS

```
HELP:   Read records from a file
PATH:   CONV.READ
TYPE:   OBJECT
SYNTAX: RECORD(FILE='str'/STREAM/DUMMY,RECMODE=STOP/CUT/WRAP,PRNCONTROL=DETACH/RETAIN/ERASE ↩
    ,SUPPADDING,ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZE=num,RECFORMAT=NONE/BIN/TXT/ ↩
    DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX- ↩
    REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR ↩
    /FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/ ↩
    VRA/VRM,RECLENGTH=num,RECDELIM='bin'/CR-ASCII/LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF- ↩
    EBCDIC/NL-EBCDIC/CRLF-EBCDIC/CR-UTF08/LF-UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE ↩
    /NL-UTF16BE/CRLF-UTF16BE/CR-UTF16LE/LF-UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF- ↩
    UTF32BE/NL-UTF32BE/CRLF-UTF32BE/CR-UTF32LE/LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE,RECCOUNT= ↩
    num,LENFORMAT.{},KEYPOSITION=num,KEYLENGTH=num,DECODE[=NONE/FLAMFILE/FIODEC/ALWAYS], ↩
    DECRYPT[{}...],FDECODE/COD.{},CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,BOM, ↩
    SUBSYSTEM(),REMOVE,LANG='str',PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read record provides a list of records from a host-like data set format. If a record contains text data, you can mark it by defining the correct CCSID. This will result in character set conversion from the provided CCSID to UTF-8 for each record. If the CCSID is not provided, the record is handled as binary data.

The following options are supported:

- Stop, cut or wrap if a record is longer then provided record length
- Print control character handling as attributes
- Padding of fix record formats at write operations
- Suppress trailing padding characters for fix record formats at read
- Suppress trailing padding characters for variable record formats at write
- All kind of record formats and length including entry, index, keyed and relative data sets, including management of slot numbers
- Support of various file allocation, space parameters and subsystems

Decryption and decompression are only supported for FLAMFILEs at record I/O because other formats like GZIP or PGP are block-oriented. For example *read.text()* can be used to convert a compressed GZIP file into records.

If the write operation is text-oriented (write.char/text/xml()) or the CCSID is defined and the file for the read operation is a FLAMFILE then the records are read via the FLAM record interface. In all other cases, the decompression of a FLAMFILE can be enforced with the keyword *DECODE*. To enforce base decoding for each record, you must define one of the corresponding switches.

For not cataloged data sets or record-oriented file formats on non record-oriented platforms (WIN, UNIX, ...), you can define file organization, block size, record format and record length. If you work with cataloged files or FLAMFILEs, these parameters are not required.

For non-record-oriented platforms, you can define a lot of different length formats. If you don't specify a length format at read and a 4 byte record length format is detected (a FILEDATA=RECORD file is allocated with FILEDATA=BINARY), the length fields in the data are used to build the records.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`
- `NUMBER:RECMODE=STOP/CUT/WRAP -Mode used to read records [STOP]`
  - `STOP -Stop if record too long`
  - `CUT -Cut if record too long`
  - `WRAP -Wrap if record too long`
- `SWITCH:SUPPADDING -Suppress trailing padding characters (useful for fix records)`
- `NUMBER:ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [catalog/SEQ]`
  - `SEQ -Sequential data set [DEFAULT]`
  - `LIB -Library (ZOS-PDSE)`
  - `PDS -Partitioned data set (ZOS-PDS)`
  - `USS -UNIX files (path based)`
  - `EDS -Entry data set (VSAM-ESDS)`
  - `KDS -Keyed data set (VSAM-KSDS)`
  - `RDS -Relative data set (VSAM-RRDS)`
- `NUMBER:RECFORMAT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format for read operation [catalog/VAR]`

- NONE -No format defined
- BIN -Open -Binary (undefined) data (only the last record are shorter then record le ngth)
- TXT -Open -Text data with standard delimiter (result is a stripped variable length r ecords)
- DLM -Open -Records separated by a special delimiter with the result of variable len gth records
- VAR -Open -Variable length record (if necessary define length format)
- VAR-ASA -Open -Variable length record with ASA print control character (if necessar y define length format)
- VAR-MCC -Open -Variable length record with machine print control codes (if necessar y define length format)
- FIX -Open -Fix length record (file size =N * record length)
- FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
- FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
- VAR-REL -Open -Variable relative record (if necessary define length format)
- VAR-REL-ASA -Open -Variable relative record with ASA print control character (if ne cessary define length format)
- VAR-REL-MCC -Open -Variable relative record with machine print control codes (if ne cessary define length format)
- FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
- FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size = N * (record length + 9))
- FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size = N * (record length + 9))
- F -Host -Fixed-length, unblocked
- FA -Host -Fixed-length, unblocked, ASA print control characters
- FM -Host -Fixed-length, unblocked, machine print control codes
- FS -Host -Fixed-length, standard
- FSA -Host -Fixed-length, standard, ASA print control characters
- FSM -Host -Fixed-length, standard, machine print control codes
- FB -Host -Fixed-length, blocked
- FBA -Host -Fixed-length, blocked, ASA print control characters
- FBM -Host -Fixed-length, blocked, machine print control codes
- FBS -Host -Fixed-length, blocked, standard
- FBSA -Host -Fixed-length, blocked, standard, ASA print control characters
- FBSM -Host -Fixed-length, blocked, standard, machine print control codes
- FE -Host -Fixed-length, ESDS
- FEA -Host -Fixed-length, ESDS, ASA print control characters
- FEM -Host -Fixed-length, ESDS, machine print control codes
- FK -Host -Fixed-length, KSDS
- FKA -Host -Fixed-length, KSDS, ASA print control characters
- FKM -Host -Fixed-length, KSDS, machine print control codes
- FR -Host -Fixed-length, RRDS

  - FRA -Host -Fixed-length, RRDS, ASA print control characters
  - FRM -Host -Fixed-length, RRDS, machine print control codes
  - U -Host -Undefined-length
  - UA -Host -Undefined-length, ASA print control characters
  - UM -Host -Undefined-length, machine print control codes
  - V -Host -Variable
  - VA -Host -Variable, ASA print control characters
  - VM -Host -Variable, machine print control codes
  - VS -Host -Variable, spanned
  - VSA -Host -Variable, spanned, ASA print control characters
  - VSM -Host -Variable, spanned, machine print control codes
  - VB -Host -Variable, blocked
  - VBA -Host -Variable, blocked, ASA print control characters
  - VBM -Host -Variable, blocked, machine print control codes
  - VBS -Host -Variable, blocked, spanned
  - VBSA -Host -Variable, blocked, spanned, ASA print control characters
  - VBSM -Host -Variable, blocked, spanned, machine print control codes
  - VE -Host -Variable, ESDS
  - VEA -Host -Variable, ESDS, ASA print control characters
  - VEM -Host -Variable, ESDS, machine print control codes
  - VK -Host -Variable, KSDS
  - VKA -Host -Variable, KSDS, ASA print control characters
  - VKM -Host -Variable, KSDS, machine print control codes
  - VR -Host -Variable, RRDS
  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes

- NUMBER:RECLENGTH=num -Maximal record length for read operation [catalog/512]
- STRING:RECDELIM='bin'/CR-ASCII/LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF-EBCDIC/NL-EBC DIC/CRLF-EBCDIC/CR-UTF08/LF-UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE/NL-UTF16BE/ CRLF-UTF16BE/CR-UTF16LE/LF-UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF-UTF32BE/NL- UTF32BE/CRLF-UTF32BE/CR-UTF32LE/LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE -Delimiter used t o define end of record (if RECF=DLM used)
  - CR-ASCII -Delimiter:carriage return in ASCII (0x0D)
  - LF-ASCII -Delimiter:line feed in ASCII (0x0A)
  - NL-ASCII -Delimiter:new line in ASCII (0x85)
  - CRLF-ASCII -Delimiter:carriage return line feed in ASCII (0x0D0A)
  - CR-EBCDIC -Delimiter:carriage return in EBCDIC (0x0D)
  - LF-EBCDIC -Delimiter:line feed in EBCDIC (0x25)
  - NL-EBCDIC -Delimiter:new line in EBCDIC (0x15)
  - CRLF-EBCDIC -Delimiter:carriage return line feed in EBCDIC (0x0D25)
  - CR-UTF08 -Delimiter:carriage return in UTF-8 (0x0D)
  - LF-UTF08 -Delimiter:line feed in UTF-8 (0x0A)
  - NL-UTF08 -Delimiter:new line in UTF-8 (0xC285)
  - CRLF-UTF08 -Delimiter:carriage return line feed in UTF-8 (0x0D0A)

  - CR-UTF16BE -Delimiter:carriage return in UTF-16BE (0x000D)
  - LF-UTF16BE -Delimiter:line feed in UTF-16BE (0x000A)
  - NL-UTF16BE -Delimiter:new line in UTF-16BE (0x0085)
  - CRLF-UTF16BE -Delimiter:carriage return line feed in UTF-16BE (0x000D000A)
  - CR-UTF16LE -Delimiter:carriage return in UTF-16LE (0x0D00)
  - LF-UTF16LE -Delimiter:line feed in UTF-16LE (0x0A00)
  - NL-UTF16LE -Delimiter:new line in UTF-16LE (0x8500)
  - CRLF-UTF16LE -Delimiter:carriage return line feed in UTF-16LE (0x0D000A00)
  - CR-UTF32BE -Delimiter:carriage return in UTF-32BE (0x0000000D)
  - LF-UTF32BE -Delimiter:line feed in UTF-32BE (0x0000000A)
  - NL-UTF32BE -Delimiter:new line in UTF-32BE (0x00000085)
  - CRLF-UTF32BE -Delimiter:carriage return line feed in UTF-32BE (0x0000000D0000000A)
  - CR-UTF32LE -Delimiter:carriage return in UTF-32LE (0x0D000000)
  - LF-UTF32LE -Delimiter:line feed in UTF-32LE (0x0A000000)
  - NL-UTF32LE -Delimiter:new line in UTF-32LE (0x85000000)
  - CRLF-UTF32LE -Delimiter:carriage return line feed in UTF-32LE (0x0D0000000A000000)
- NUMBER:RECCOUNT=num -Amount of records handled in one read operation [catalog/128]
- NUMBER:KEYPOSITION=num -Key position (>=1) for index/key sequential data sets [0=NONE]
- NUMBER:KEYLENGTH=num -Key length (>=1) for index/key sequential data sets [if KEYPOS >0 then 8]
- NUMBER:DECODE=NONE/FLAMFILE/FIODEC/ALWAYS -Decode encoded, encrypted or compressed d ata [0=NONE]
  - NONE -No automated decoding
  - FLAMFILE -Transparent read of FLAMFILEs
  - FIODEC -Decoding after file IO (or read of FLAMFILE)
  - ALWAYS -Always decode encoded data (causes the same as FIODEC)
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.2.59 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   CONV.READ.RECORD
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.2.60 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size for read operations [catalog/AUTO]
PATH:   CONV.READ.RECORD
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.2.61 OVERLAY *LENFORMAT*

SYNOPSIS

```
HELP:   Format of the length field for open platforms [word]
PATH:   CONV.READ.RECORD
TYPE:   OVERLAY
SYNTAX: LENFORMAT.{INTEGER()/STRING()/BCD()/HOST()}
```

DESCRIPTION The overlay can be used to define the format of the length filed on open platforms which don't support record formats by the data management system. For example: If you read records on a mainframe and you want to write the records as records (not as binary or text data) on WINDOWS or UNIX then the data will be prefixed with the record length. The format of this length field can be changed over this union.

This length specification will be only used for variable length records. The default mapping method between record formats will be map each host record format the corresponding open variable format.

If the length format not defined the length field will be written as 32 bit integer in platform dependent byte order. This behavior corresponds to the FILEDATA=RECORD parameter for USS files on z/OS. Means if you write a record oriented file to USS without any format changed over this overlay then you can read the record with another MVS application using FILEDATA=RECORD.

If you choose record format text or binary then this will correspond to FILEDATA=TEXT or FILEDATA=BINARY and the length format specification will be ignored. To enable record format mapping based on the data type you can set the environment variable below:

```
FL_RECORD_FORMAT_MAPPING=DATATYPE
```

If this environment variable set, then the length format is only used if you explicitly set a variable record format of the open systems.

Since version 5.1.8, FLAM can detect the 4 byte length fields (integer or host) inside a data stream. We recommend to use one of these formats to have more comfort when reading.

4.2.62 OBJECT *INTEGER*

SYNOPSIS

```
HELP:   Use integer format for the length field
PATH:   CONV.READ.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: INTEGER(ENDIAN=SYSTEM/BIG/LITTLE,WIDTH=U16/U32/U64,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in integer format. You can define the byte order, the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [SYSTEM]
  - SYSTEM -Byte order of the current system
  - BIG -Big endian byte order
  - LITTLE -Little endian byte order
- NUMBER:WIDTH=U16/U32/U64 -Bit width/rate [B32]
  - U16 -16 bit unsigned integer
  - U32 -32 bit unsigned integer
  - U64 -64 bit unsigned integer
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.2.63 OBJECT *STRING*

SYNOPSIS

```
HELP:   Use string format for the length field
PATH:   CONV.READ.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: STRING(CHRSET=SYSTEM/ASCII/EBCDIC,COUNT=C04/C05/C06,BASE=B10,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in string format (a character per digit). You can define the character set, the character count, the base and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:CHRSET=SYSTEM/ASCII/EBCDIC -Character set [SYSTEM]
  - SYSTEM -Use system code page
  - ASCII -Use ASCII (UTF-8) code page
  - EBCDIC -Use EBCDIC code page
- NUMBER:COUNT=C04/C05/C06 -Character count [C06]
  - C04 -4 character/digits
  - C05 -5 character/digits
  - C06 -6 character/digits
- NUMBER:BASE=B10 -String base [B10]
  - B10 -Base 10 (decimal)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.2.64 OBJECT *BCD*

SYNOPSIS

```
HELP:   Use BCD format for the length field
PATH:   CONV.READ.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: BCD(WIDTH=U16/U24/U32,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in BCD format (4 Bit are one decimal digit). You can define the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:WIDTH=U16/U24/U32 -Bit width/rate [B32]
  - U16 -16 bit unsigned BCD (POV with 4 digits)
  - U24 -24 bit unsigned BCD (POV with 6 digits)
  - U32 -32 bit unsigned BCD (POV with 8 digits)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.2.65 OBJECT *HOST*

SYNOPSIS

```
HELP:   Use host format (LLxx) for the length field
PATH:   CONV.READ.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: HOST(ENDIAN=SYSTEM/BIG/LITTLE,EXCLUDE)
```

DESCRIPTION The object defines the 4 byte host specific format (LLxx) of the length field. This format uses the first 16 bits as integer in big endian or little endian byte order to represent the record length including or excluding the record length field itself and the last 16 bits are set to zero when writing and ignored when reading.

For example, a record with 136 bytes content has a default length field (big endian, inclusive) of:

```
x'008D0000'
```

The default conforms to the length fields used by the data management system of MVS.

ARGUMENTS

- NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [BIG ENDIAN] for first 16 Bit
  - SYSTEM -Byte order of the current system
  - BIG -Big endian byte order
  - LITTLE -Little endian byte order
- SWITCH:EXCLUDE -Record length field not part of the length value [OFF]

4.2.66 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   CONV.READ.RECORD
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.2.67 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   CONV.READ.RECORD.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']
  - DEFAULT -FLAM4 default password

4.2.68 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   CONV.READ.RECORD.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- STRING:LIBRARY='str' -Library name for FKME ['libfkme']
- STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']
- STRING:PARAMETER='str' -Parameter for the call of FKME [”]

4.2.69 OVERLAY *FDECODE*

SYNOPSIS

```
HELP:   Additional enforced decoding [NONE]
PATH:   CONV.READ.RECORD
TYPE:   OVERLAY
SYNTAX: FDECODE/COD.{BASE64/OPGP/BASE32/BASE16}
```

DESCRIPTION The overlay *fdecode* allows you to enforce an additional decoding of the data after the auto decoding (*decode* parameter defined) was done. This is useful if the resulting data is still encoded. This may happen if a record was decoded from a FLAMFILE (read.record()) and the record is, for example, base encoded.

The forced decoding works on records and gives you the possibility to read files, which (for example) were encoded with:

```
'write.record(encode.base64())'
```

The enforcement of base decoding can even be used if no auto decoding is activated.

See below for the decoding possibilities.

ARGUMENTS

- `SWITCH:BASE64/OPGP -Base64 decoding (RFC4648)`
- `SWITCH:BASE32 -Base32 decoding (RFC4648)`
- `SWITCH:BASE16 -Base16 decoding (RFC4648)`

4.2.70 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID (enables text handling if set)
PATH:   CONV.READ.RECORD
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
   flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (auto-detect)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`
- `BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)`
- `BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)`

4.2.71 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   CONV.READ.RECORD
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.2.72 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   CONV.READ.RECORD
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
SUBSYS(NAME='name' parm='keyword,keyword=value,value')
SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.2.73 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   CONV.READ.RECORD
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.2.74 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   CONV.READ.RECORD
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- WIN -Choose Windows as platform (CRLF)
- UNX -Choose Unix derivative as platform (LF)
- ZOS -Choose ZOS as platform (NL)
- USS -Choose USS of ZOS as platform (NL)
- VSE -Choose VSE as platform (NL)
- BS2 -Choose BS2000-OSD as platform (NL)
- MAC -Choose MAC as platform (CR)

4.2.75 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   CONV.READ.RECORD
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)

  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.2.76 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   CONV.READ.RECORD.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.2.77 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   CONV.READ.RECORD
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.2.78 OBJECT *FLAM4*

SYNOPSIS

```
HELP:   Read records from a FLAM4FILE
PATH:   CONV.READ
TYPE:   OBJECT
SYNTAX: FLAM4(FILE='str',MEMBER='str',PRNCONTROL=DETACH/RETAIN/ERASE,RECCOUNT=num,RECLENGTH ↩
    =num,TRUNCATE,PASSWORD/CRYPTOKEY='bin'/DEFAULT,KMLIBRARY='str',KMFUNCTION='str', ↩
    KMPARAMETER='str',CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,BOM,REMOVE,LANG='str', ↩
    PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH())
```

DESCRIPTION Read FLAM4 works with a list of records from a FLAMFILE of version 4 or older. If such a record is text data, you can mark this by defining the correct CCSID. Such a definition will result in character set conversion from the provided CCSID to UTF-8 for each record. If no CCSID is provided, the record will be handled as binary data.

For FLAMFILE archives, these features are supported:

- Read all members or a dedicated member of the archive
- Print control character handling defined by a dedicated mode
- Pass phrase based or FKME based decryption technology
- Free definition of the amount of records handled as one chunk

User I/O, pre- and post-processing exits are not supported. If these features are required, please use the extended variant (XCM-P/XCNV) of this command.

If you need, for example, Base64 decoding for records of a FLAMFILE, please use read.record(fdecode.base64). Read.record() can transparently read FLAMFILEs, supports base decoding and other useful features for record handling.

ARGUMENTS

- `STRING:FILE='str' -Name/URL of the FLAMFILE to read [required]`
- `STRING:MEMBER='str' -Name or Index of the member in the FLAMFILE to read`
- `NUMBER:RECCOUNT=num -Amount of records handled in one read operation [128]`
- `NUMBER:RECLENGTH=num -Maximal record length for read operation [512]`
- `SWITCH:TRUNCATE -Allow truncation of records [FALSE]`

- STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to decrypt the FLAMFILE [”]
  - DEFAULT -FLAM4 default password
- STRING:KMLIBRARY='str' -Library name for FKME [”]
- STRING:KMFUNCTION='str' -Function name of FKME [”]
- STRING:KMPARAMETER='str' -Parameter for the call of FKME [”]
- SWITCH:REMOVE -Remove FLAMFILE after successful processing [FALSE]

4.2.79 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   CONV.READ.FLAM4
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- DETACH -Manage print control character as record attribute
- RETAIN -Leave print control character as part of the data record
- ERASE -Erase the print control character from the record

4.2.80 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID (enables text handling if set)
PATH:   CONV.READ.FLAM4
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)

- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (auto-detect)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`
- `BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)`
- `BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)`

4.2.81 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   CONV.READ.FLAM4
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.2.82 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   CONV.READ.FLAM4
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.2.83 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   CONV.READ.FLAM4
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- WIN -Choose Windows as platform (CRLF)
- UNX -Choose Unix derivative as platform (LF)
- ZOS -Choose ZOS as platform (NL)
- USS -Choose USS of ZOS as platform (NL)
- VSE -Choose VSE as platform (NL)
- BS2 -Choose BS2000-OSD as platform (NL)
- MAC -Choose MAC as platform (CR)

4.2.84 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   CONV.READ.FLAM4
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)

  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.2.85 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   CONV.READ.FLAM4.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                              ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'                  ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'                 ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                    ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'         ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.2.86 OBJECT *AUTO*

SYNOPSIS

```
HELP:   Read binary, text or XML data record- or block-oriented
PATH:   CONV.READ
TYPE:   OBJECT
SYNTAX: AUTO(FILE='str'/STREAM/DUMMY,BLKSIZE=num,RECLENGTH=num,RPLFFD=num,CCSID='str'/ ↩
    DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,BOM,ENL2LF,DECRYPT[{}...],REMOVE,LANG='str',PLATFORM= ↩
    WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read *auto* attempts to identify the file format automatically and reads the file with the most appropriate parameters. Detection works on a best effort basis and should not be relied on.

For example, "read.auto() write.record()" can convert every supported file format to records on mainframe systems. This is utilized in the FLVIEW and FLVEDIT ISPF command.

With read.auto(), you can transparently read normal XML or text files with delimiters, encoded (BASE16/32/64), encrypted (FLAMFILE) or compressed (GZIP, BZIP2, XZ(LZMA)) files (even nested!), normal record-oriented data sets (FB(A/M)/VB(A/M)/VSA FLAMFILEs or binary data. If the data contains 4 byte length fields after I/O, decryption, decompression then a list of records is built to form one block. This allows you for example to read a file transparently from USS which was created with FILEDATA=RECORD and now allocated with FILEDATA=BINARY or a record length based format from a ZIP archive. The read.auto() function was mainly implemented to display (FLVIEW) the content of a file. The detection may fail on some files, especially for very small files.

Generally: If you know the type of data (binary, char, text, xml, flam, charset/ccsid), it is recommended to use the corresponding read method and set the known parameters to prevent false detections. On the other side, if you know nothing about your data or you get data from different sources, read.auto() might be very helpful.

To simplify the use of the read.auto() function, the mechanisms below are implemented:

- The EBCDIC new line to ASCII line feed management is activated. This means that, for methods that support it, the ENL2LF flag is set for read methods and the ELF2NL flag is set for the write methods.
- Suppression of trailing whitespace is activated for text records.
- The print control character handling is set to erase print control characters from records.
- Subsystem specifications are not supported
- Base encodings are always decoded, if the decoding results in useful data. (i.e. text files in an a Base64 encoded Gzip file are decoded and uncompressed automatically). Optional CRC verification and verification of the trailer (if ASCII armor) is done if the CHECK switch is enabled.
- Record length format detection is enabled and used after I/O (a wrong static file allocation on z/OS was done (FILEDATA=BINARY instead of FILEDATA=RECORD) or a file with variable record length was read on Windows or UNIX platforms), after decryption (data containing record length fields were encrypted), after decompression (data with record length fields was compressed (ZIP)) and after base decoding (if data with length fields was encode).

To read ASCII text files with NEL delimiter or to use specific formatting options, please use read.text(). To keep and manage record attributes, please use read.record(). The read.auto() function is designed to be very powerful, using the procedures suited best and, at the same time, to keep it as simple as possible.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`
- `NUMBER:RECLENGTH=num -Maximum record length (Host) /line length (Unix/Win) for text parsing [512]`

- NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assumin g n lines per page [0 =no replacement]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.2.87 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size to read [AUTO]
PATH:   CONV.READ.AUTO
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.2.88 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID [auto detection]
PATH:   CONV.READ.AUTO
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE

- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (auto-detect)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`
- `BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)`
- `BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)`

4.2.89 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   CONV.READ.AUTO
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.2.90 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   CONV.READ.AUTO
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.2.91 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   CONV.READ.AUTO
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.2.92 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   CONV.READ.AUTO.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
 * In hexadecimal format (recommended): password=x'hex'
 * In ASCII format: password=a'ascii'
 * In EBCDIC format: password=e'ebcdic'
 * In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']`
  - `DEFAULT -FLAM4 default password`

4.2.93 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   CONV.READ.AUTO.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- STRING:LIBRARY='str' -Library name for FKME ['libfkme']
- STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']
- STRING:PARAMETER='str' -Parameter for the call of FKME [”]

4.2.94 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   CONV.READ.AUTO
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.2.95 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   CONV.READ.AUTO
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- WIN -Choose Windows as platform (CRLF)
- UNX -Choose Unix derivative as platform (LF)
- ZOS -Choose ZOS as platform (NL)

- USS -Choose USS of ZOS as platform (NL)
- VSE -Choose VSE as platform (NL)
- BS2 -Choose BS2000-OSD as platform (NL)
- MAC -Choose MAC as platform (CR)

4.2.96 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   CONV.READ.AUTO
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)

  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.2.97 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   CONV.READ.AUTO.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.2.98 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   CONV.READ.AUTO
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.2.99 PARAMETER *FILE*

SYNOPSIS

```
HELP:   Abbreviation for read.auto(file=...)
PATH:   CONV.READ
TYPE:   STRING
SYNTAX: FILE='str'/STREAM/DUMMY
```

DESCRIPTION `read.file='name.txt'` is an abbreviation for `read.auto(file='name.txt')` without any further parameters. Please have a look at read.auto() for more information.

`read.file` use all data format detection capabilities of FLAM to read transparent byte streams or records form a data source, which can be an encoded (e.g. Base64), encrypted (e.g. PGP) and/or compressed (GZIP/BZIP2/XZ) file, a FLAMFILE, other archive or a normal clear data set.

The data stream after I/O and after each conversion can contain several kinds of 4 byte length field formats. This is also detected and the data is interpreted as blocks containing various records for the next layer.

SELECTIONS

- `STREAM -Read from stdin or write to stdout`
- `DUMMY -Read EOF or write nothing`

4.2.100 OVERLAY *WRITE*

SYNOPSIS

```
HELP:   Outbound parameter to write data
PATH:   CONV
TYPE:   OVERLAY
SYNTAX: WRITE.{BINARY()/CHARACTER()/TEXT()/XML()/RECORD()/FLAM4()}
```

DESCRIPTION Over the overlay WRITE you can choose the target format (binary, text, character, record, FLAM4FILE, table, ...) which will be written. Depending of this target a dedicated behavior is implemented and the corresponding parameter can be used to detail the procedure. Based on the knowledge about the kind of data You have read and the format you want to write, we have simplify the way to do automatically the right conversions for You.

Additionally you can calculate or verify checksums on the binary data stream direct in front of the write operation.

Below You can find the currently supported targets explained in a dedicated chapter.

4.2.101 OBJECT *BINARY*

SYNOPSIS

```
HELP:   Write binary data to a file
PATH:   CONV.WRITE
TYPE:   OBJECT
SYNTAX: BINARY(FILE='str'/STREAM/DUMMY,FALLOC(),REMAIN,APPEND,FLUSH,METHOD=L4I/L4X/B4I/B4X/ ↩
    S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX,COMPRESS/CMP.{},ENCRYPT/ENC.{},ENCODE/COD.{},FLAM4NDC, ↩
    PRNCONTROL=DETACH/RETAIN/ERASE,FRCREC,PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH())
```

DESCRIPTION Write binary formats all elements to one block, this binary block can then optionally compressed, encrypted and encoded and is written or appended to the specified file. For compression, encryption and encoding one of the supported methods can be used. For each of these methods specific parameters can be set to control the out bound conversions.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to write ["==origin.ext]`

  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

- SWITCH:REMAIN -Remain existing files (don't overwrite) [FALSE]
- SWITCH:APPEND -Append data to this file [FALSE]
- SWITCH:FLUSH -Enforce flush in front of close file [FALSE]
- NUMBER:METHOD=L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX -Method to convert records in blocks [NONE]
  - L4I -Adds 4 byte length fields:Little endian integer, length inclusive
  - L4X -Adds 4 byte length fields:Little endian integer, length exclusive (ZIP)
  - B4I -Adds 4 byte length fields:Big endian integer, length inclusive
  - B4X -Adds 4 byte length fields:Big endian integer, length exclusive (USS)
  - S4I -Adds 4 byte length fields:System endian integer, length inclusive
  - S4X -Adds 4 byte length fields:System endian integer, length exclusive (VAR)
  - HLI -Adds 4 byte length fields:Little endian short (LL00), length inclusive
  - HLX -Adds 4 byte length fields:Little endian short (LL00), length exclusive
  - HBI -Adds 4 byte length fields:Big endian short (LL00), length inclusive (MVS)
  - HBX -Adds 4 byte length fields:Big endian short (LL00), length exclusive
  - HSI -Adds 4 byte length fields:System endian short (LL00), length inclusive
  - HSX -Adds 4 byte length fields:System endian short (LL00), length exclusive
- SWITCH:FLAM4NDC -Write data as FLAM4FILE in MODE=NDC(No Data Compression) [FALSE]
- SWITCH:FRCREC -Enforce record orientation on record oriented devices [FALSE]

4.2.102 OBJECT *FALLOC*

SYNOPSIS

```
HELP:   Platform dependent file allocation parameter
PATH:   CONV.WRITE.BINARY
TYPE:   OBJECT
SYNTAX: FALLOC(ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZE=num,RECFORMAT=NONE/BIN/TXT/ ↩
    DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX- ↩
    REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR ↩
    /FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/ ↩
    VRA/VRM,RECLENGTH=num,KEYDISP=OLD/NEW/DEL,KEYPOSITION=num,KEYLENGTH=num,SUBSYSTEM(), ↩
    SPACE(),LIKE='str',RENEW)
```

DESCRIPTION The file allocation object collects several parameters for platforms which are record oriented based on a catalog system like z/OS® and where the space, volume and other things must be predefined. The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

But also on distributed platforms like (WIN, UNIX, MAC, ...) FLAM supports record oriented file types to keep the record information on such platforms. For this case the different record format for the open world can be defined. To read such a special file format on an open platform the correct record format must also be defined. On a catalog based system, this is not required. If the record format in the catalog are not the same as specified an error will occur.

ARGUMENTS

- NUMBER:ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [SEQ]

  - SEQ -Sequential data set [DEFAULT]
  - LIB -Library (ZOS-PDSE)
  - PDS -Partitioned data set (ZOS-PDS)
  - USS -UNIX files (path based)
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)
- NUMBER:BLKSIZE=num -Block size [system]
- NUMBER:RECFORMAT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format [auto]
  - NONE -No format defined
  - BIN -Open -Binary (undefined) data (only the last record are shorter then record length)
  - TXT -Open -Text data with standard delimiter (result is a stripped variable length records)
  - DLM -Open -Records separated by a special delimiter with the result of variable length records
  - VAR -Open -Variable length record (if necessary define length format)
  - VAR-ASA -Open -Variable length record with ASA print control character (if necessary define length format)
  - VAR-MCC -Open -Variable length record with machine print control codes (if necessary define length format)
  - FIX -Open -Fix length record (file size =N * record length)
  - FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
  - FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
  - VAR-REL -Open -Variable relative record (if necessary define length format)
  - VAR-REL-ASA -Open -Variable relative record with ASA print control character (if necessary define length format)
  - VAR-REL-MCC -Open -Variable relative record with machine print control codes (if necessary define length format)
  - FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
  - FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size =N * (record length + 9))
  - FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size =N * (record length + 9))
  - F -Host -Fixed-length, unblocked
  - FA -Host -Fixed-length, unblocked, ASA print control characters
  - FM -Host -Fixed-length, unblocked, machine print control codes
  - FS -Host -Fixed-length, standard
  - FSA -Host -Fixed-length, standard, ASA print control characters
  - FSM -Host -Fixed-length, standard, machine print control codes
  - FB -Host -Fixed-length, blocked

  - FBA -Host -Fixed-length, blocked, ASA print control characters
  - FBM -Host -Fixed-length, blocked, machine print control codes
  - FBS -Host -Fixed-length, blocked, standard
  - FBSA -Host -Fixed-length, blocked, standard, ASA print control characters
  - FBSM -Host -Fixed-length, blocked, standard, machine print control codes
  - FE -Host -Fixed-length, ESDS
  - FEA -Host -Fixed-length, ESDS, ASA print control characters
  - FEM -Host -Fixed-length, ESDS, machine print control codes
  - FK -Host -Fixed-length, KSDS
  - FKA -Host -Fixed-length, KSDS, ASA print control characters
  - FKM -Host -Fixed-length, KSDS, machine print control codes
  - FR -Host -Fixed-length, RRDS
  - FRA -Host -Fixed-length, RRDS, ASA print control characters
  - FRM -Host -Fixed-length, RRDS, machine print control codes
  - U -Host -Undefined-length
  - UA -Host -Undefined-length, ASA print control characters
  - UM -Host -Undefined-length, machine print control codes
  - V -Host -Variable
  - VA -Host -Variable, ASA print control characters
  - VM -Host -Variable, machine print control codes
  - VS -Host -Variable, spanned
  - VSA -Host -Variable, spanned, ASA print control characters
  - VSM -Host -Variable, spanned, machine print control codes
  - VB -Host -Variable, blocked
  - VBA -Host -Variable, blocked, ASA print control characters
  - VBM -Host -Variable, blocked, machine print control codes
  - VBS -Host -Variable, blocked, spanned
  - VBSA -Host -Variable, blocked, spanned, ASA print control characters
  - VBSM -Host -Variable, blocked, spanned, machine print control codes
  - VE -Host -Variable, ESDS
  - VEA -Host -Variable, ESDS, ASA print control characters
  - VEM -Host -Variable, ESDS, machine print control codes
  - VK -Host -Variable, KSDS
  - VKA -Host -Variable, KSDS, ASA print control characters
  - VKM -Host -Variable, KSDS, machine print control codes
  - VR -Host -Variable, RRDS
  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes

- NUMBER:RECLENGTH=num -Record length (data only, no print control and length fields)
- NUMBER:KEYDISP=OLD/NEW/DEL -Key disposition for index/key sequential data sets [OLD]
  - OLD -Key disposition OLD (keep the existing key)
  - NEW -Key disposition NEW (insert record number as key)
  - DEL -Key disposition DEL (delete key from record)

- NUMBER:KEYPOSITION=num -Key position (>=1) for index/key sequential data sets [1]
- NUMBER:KEYLENGTH=num -Key length (>=1) for index/key sequential data sets [8]
- STRING:LIKE='str' -Specifies the name of the model data set from which the attributes are to be copied
- SWITCH:RENEW -Enforce new allocation by a remove of an existing file

4.2.103 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   CONV.WRITE.BINARY.FALLOC
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- STRING:NAME='str' -Name of the subsystem
- STRING:PARM='str' -Parameter list for the subsystem (comma separated)

4.2.104 OBJECT *SPACE*

SYNOPSIS

```
HELP:   Platform dependent space definition parameter
PATH:   CONV.WRITE.BINARY.FALLOC
TYPE:   OBJECT
SYNTAX: SPACE(PRIMARY=num,SECONDARY=num,DIRBLOCKS=num,VOLSER='str',VOLUNIT='str',DATACLASS ↩
    ='str',MGMTCLASS='str',STORCLASS='str',NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG, ↩
    ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG,RELEASE=YES/NO,UNMOVABLE,CONTIG,PERMANENT,RLS= ↩
    NRI/CR/CRE)
```

DESCRIPTION The space definition object collects several parameters for platforms where data sets are preallocated like z/OS® and where clylinders or tracks must be predefined for a file allocation.

The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

All size values must be given in megabyte (MiB). The calculation in the platform dependent units (tracks, cylinders, ...) are done inernally. Note: 1 MiB = 1024 KiB = 1024*1024 Byte

ARGUMENTS

- NUMBER:PRIMARY=num -Primary size in megabytes
- NUMBER:SECONDARY=num -Secondary size in megabytes
- NUMBER:DIRBLOCKS=num -Directory blocks for partitioned data sets
- STRING:VOLSER='str' -Volume serial number
- STRING:VOLUNIT='str' -Volume unit
- STRING:DATACLASS='str' -Data class
- STRING:MGMTCLASS='str' -Management class
- STRING:STORCLASS='str' -Storage class
- NUMBER:NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for normal end [CATALOG]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for abnormal end [DELETE]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:RELEASE=YES/NO -Release unused space when file is closed [SEQ=YES, PDS=NO]
  - YES -Yes, do it
  - NO -No, don't do it
- SWITCH:UNMOVABLE -Allocate space unmovable [FALSE]
- SWITCH:CONTIG -Allocate space contiguously [FALSE]
- SWITCH:PERMANENT -Allocate space permanently [FALSE]
- NUMBER:RLS=NRI/CR/CRE -Specifies the type of record level sharing (RLS) being done for a specific data set [NONE]
  - NRI -No Read Integrity
  - CR -Consistent Read
  - CRE -Consistent Read Explicit

4.2.105 OVERLAY *COMPRESS*

SYNOPSIS

```
HELP:   Compression procedure [NONE]
PATH:   CONV.WRITE.BINARY
TYPE:   OVERLAY
SYNTAX: COMPRESS/CMP.{GZIP()/BZIP2()/XZ()/FLAM4()}
```

DESCRIPTION With the overlay compress you can choose which kind of compression method is used for the data. Depending on the compression method different parameter can be provided to define the behavior in more detail.

If no compression method selected no compression of the data will be done.

See below the different possibilities of compression.

4.2.106 OBJECT *GZIP*

SYNOPSIS

```
HELP:   Compress data compliant to RCF1952 (GZIP)
PATH:   CONV.WRITE.BINARY.COMPRESS
TYPE:   OBJECT
SYNTAX: GZIP(LEVEL=num/COPY/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO, ↩
    COMMENT='str',CHECK)
```

DESCRIPTION The GZIP compression conforms to RCF1951/52. The compression level can be set to values from 0 to 9 (0=copy, 1=fastest, 9=best). Additionally, it is possible to set a comment and to enforce checksum calculation of the file header.

The checksum calculation is not recommended because only the newer versions of Zlib can handle this checksum technique. With older versions of Zlib this can cause problems. Therefore, checksum calculation is turned off by default.

The decompressor also accepts the older Zlib wrapping format specified in RFC1950. It is not possible to enforce RFC1950 at compression because most standards require RFC1952 for the deflate algorithms.

If no compression level is defined and the data is marked is being without redundancy, level 0 (copy) is used. In all other cases the default level 6 is used.

Decompression of concatenated GZIP files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

`filename/:memberindex`

ARGUMENTS

- `NUMBER:LEVEL=num/COPY/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level [AUTO==6]`
  - `COPY -No compression =copy of data (0)`
  - `FAST -Fastest compression (level 1)`
  - `LEV1 -Compression level 1`
  - `LEV2 -Compression level 2`
  - `LEV3 -Compression level 3`
  - `LEV4 -Compression level 4`
  - `LEV5 -Compression level 5`
  - `LEV6 -Compression level 6`
  - `LEV7 -Compression level 7`
  - `LEV8 -Compression level 8`
  - `LEV9 -Compression level 9`
  - `BEST -Best compression (level 9)`
  - `AUTO -Automatic compression (level 6)`
- `STRING:COMMENT='str' -Comment for header [NONE]`
- `SWITCH:CHECK -Calculate checksum over GZIP header [FALSE]`

4.2.107 OBJECT *BZIP2*

SYNOPSIS

```
HELP:   BZIP2 data compression technique
PATH:   CONV.WRITE.BINARY.COMPRESS
TYPE:   OBJECT
SYNTAX: BZIP2(LEVEL=num/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO)
```

DESCRIPTION The BZIP2 algorithm compresses chunks of data. Such a chunk can be up to 100KB, 200KB, . . . , 900KB. The chunk size is selected automatically based on the internal block size (block size in KB rounded up to the next possible chunk size).

The chunk size directly correlates with the compression level. Compression levels range from 1 (fastest) to 9 (best, but slowest). Level 1 results in 100KB chunk size, level 9 in 900KB chunk size (i.e. chunk size = level*100KB). Hence, a lower level also reduces the required memory resources during compression and decompression.

Decompression of concatenated BZIP2 files is also possible. All concatenated files will be decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
```

ARGUMENTS

- `NUMBER:LEVEL=num/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level (defines block size) [AUTO]`
  - `FAST -Fastest compression (Block size =100kb)`
  - `LEV1 -Compression level 1 (Block size =100kb)`
  - `LEV2 -Compression level 2 (Block size =200kb)`
  - `LEV3 -Compression level 3 (Block size =300kb)`
  - `LEV4 -Compression level 4 (Block size =400kb)`
  - `LEV5 -Compression level 5 (Block size =500kb)`
  - `LEV6 -Compression level 6 (Block size =600kb)`
  - `LEV7 -Compression level 7 (Block size =700kb)`
  - `LEV8 -Compression level 8 (Block size =800kb)`
  - `LEV9 -Compression level 9 (Block size =900kb)`
  - `BEST -Best compression (Block size =900kb)`
  - `AUTO -Automatic compression (depending on the real block size)`

4.2.108 OBJECT *XZ*

SYNOPSIS

```
HELP:   XZ (LZMA) data compression technique
PATH:   CONV.WRITE.BINARY.COMPRESS
TYPE:   OBJECT
SYNTAX: XZ(LEVEL=num/FAST/LEV0/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO)
```

DESCRIPTION The XZ compression conforms to the XZ-compression tool on UNIX-systems. The compression level can be set to values from 0 to 9 (0=fastest, 9=best).

The default level is 6 when no level is specified.

Decompression of concatenated XZ files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
```

ARGUMENTS

- `NUMBER:LEVEL=num/FAST/LEV0/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level [AUTO==6]`
  - `FAST -Fastest compression (level 0)`

  - LEV0 -Compression level 0
  - LEV1 -Compression level 1
  - LEV2 -Compression level 2
  - LEV3 -Compression level 3
  - LEV4 -Compression level 4
  - LEV5 -Compression level 5
  - LEV6 -Compression level 6
  - LEV7 -Compression level 7
  - LEV8 -Compression level 8
  - LEV9 -Compression level 9
  - BEST -Best compression (level 9)
  - AUTO -Automatic compression (level 6)

4.2.109 OBJECT *FLAM4*

SYNOPSIS

```
HELP:   Compress data with FLAM
PATH:   CONV.WRITE.BINARY.COMPRESS
TYPE:   OBJECT
SYNTAX: FLAM4(MODE=CX7/CX8/VR8/ADC/NDC,COMMENT='str')
```

DESCRIPTION The FLAM4 compression to write compressed data into FLAMFILEs. As main argument you can define the compression mode (ADC, VR8, CX8). Additionally, it is possible to set a comment in the file header.

If the record-oriented method (write.record()) is used, the records with all attributes are written to the FLAM archive. If a block-oriented method (write.binary/char/text/xml()) is used, the data blocks are wrapped into records and written to the FLAMFILE as binary chunks with an undefined record format (U).

ARGUMENTS

- NUMBER:MODE=CX7/CX8/VR8/ADC/NDC -Compression mode [ADC]
  - CX7 -Compression encoding as printable characters
  - CX8 -Compression encoding as binary data
  - VR8 -Enhanced compression encoding as binary data
  - ADC -Advanced Data Compression
  - NDC -No data compression
- STRING:COMMENT='str' -Comment for the member in the FLAMFILE [internal use]

4.2.110 OVERLAY *ENCRYPT*

SYNOPSIS

```
HELP:   Encryption procedure [NONE]
PATH:   CONV.WRITE.BINARY
TYPE:   OVERLAY
SYNTAX: ENCRYPT/ENC.{F4PWD()/F4KME()}
```

DESCRIPTION The overlay *encrypt* allows you to choose which kind of encryption method is used for the data. Depending on the encryption method, different parameters can be set.

If no encryption method is selected, then no data encryption will be done.

See below for the encryption possibilities.

4.2.111 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password encryption
PATH:   CONV.WRITE.BINARY.ENCRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(MODE=AES/FLAM,PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Activates FLAM4 password-based encryption to encrypt the data written to FLAMFILEs. You can choose the encryption mode. The default and recommended mode is AES. If you provide no password, a hardcoded default password is used.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

If the data is already compressed, then MODE=NDC (no data compression) is used. Otherwise, ADC compression is used. If you set the compression mode to CX7/CX8/VR8 and enable encryption, e.g.

```
  comp.flam(mode=CX7/CX8/VR8) encr.f4pwd()
```

the compression mode is ignored and ADC/NDC is used instead. Due to how those three old compression modes work, they are incompatible with FLAM encryption.

ARGUMENTS

- NUMBER:MODE=AES/FLAM -Encryption mode [AES]
  - AES -AES encryption
  - FLAM -FLAM encryption
- STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['******']
  - DEFAULT -FLAM4 default password

4.2.112 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   CONV.WRITE.BINARY.ENCRYPT
TYPE:   OBJECT
SYNTAX: F4KME(MODE=AES/FLAM,LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 encryption of data written to FLAMFILEs based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME parameter is provided (encr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using encr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

You may define the encryption mode. The default and recommended mode is AES.

If the data is already compressed, then MODE=NDC (no data compression) is used. Otherwise, ADC compression is used to create the FLAMFILE. If you set the compression mode to CX7/CX8/VR8 and enable encryption, e.g.

```
comp.flam(mode=CX7/CX8/VR8) encr.f4kme()
```

the compression mode is ignored and ADC/NDC is used instead. Due to how those three old compression modes work, they are incompatible with FLAM encryption.

ARGUMENTS

- `NUMBER:MODE=AES/FLAM -Encryption mode [AES]`
  - `AES -AES encryption`
  - `FLAM -FLAM encryption`
- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME [”]`

4.2.113 OVERLAY *ENCODE*

SYNOPSIS

```
HELP:   Encoding procedure [NONE]
PATH:   CONV.WRITE.BINARY
TYPE:   OVERLAY
SYNTAX: ENCODE/COD.{BASE64()/BASE32()/BASE16()/OPGP()}
```

DESCRIPTION The overlay *encode* allows to choose which kind of encoding method is used for the data. Depending on the encoding method, different parameters can be set.

If no encoding method is selected, then no data encoding will be done.

See below for the available encoding options.

4.2.114 OBJECT *BASE64*

SYNOPSIS

```
HELP:   Base64 encoding (RFC4648)
PATH:   CONV.WRITE.BINARY.ENCODE
TYPE:   OBJECT
SYNTAX: BASE64(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)

  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.115 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.BINARY.ENCODE.BASE64
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.116 OBJECT *BASE32*

SYNOPSIS

```
HELP:   Base32 encoding (RFC4648)
PATH:   CONV.WRITE.BINARY.ENCODE
TYPE:   OBJECT
SYNTAX: BASE32(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.117 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.BINARY.ENCODE.BASE32
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
```

```
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.118 OBJECT *BASE16*

SYNOPSIS

```
HELP:   Base16 encoding (RFC4648)
PATH:   CONV.WRITE.BINARY.ENCODE
TYPE:   OBJECT
SYNTAX: BASE16(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)

  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)

- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.119 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.BINARY.ENCODE.BASE16
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.120 OBJECT *OPGP*

SYNOPSIS

```
HELP:   Base64 encoding with 76 character, CRC checksum and Armor clasp (RFC4880)
PATH:   CONV.WRITE.BINARY.ENCODE
TYPE:   OBJECT
SYNTAX: OPGP(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE ↩
    /UTF32BE/UCS4LE/UTF32LE,DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM)
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- `NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]`
  - `NONE -No character set defined`
  - `SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)`
  - `ASCII -ASCII (mainly used in the for open system)`
  - `UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)`
  - `UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)`
  - `EBCDIC -EBCDIC (mainly used on IBM mainframe)`
  - `UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)`
  - `UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)`
  - `UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)`
  - `UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)`
  - `UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)`

  - UTF32BE –UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE –UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE –UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM –Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST –Adds no delimiter (HOST)
  - BIN –Adds no delimiter (HOST)
  - NL –Adds delimiter new line (USS)
  - USS –Adds delimiter for USS (new line)
  - LF –Adds delimiter line feed (UNIX)
  - UNIX –Adds delimiter for UNIX (line feed)
  - CR –Adds delimiter carriage return (MAC)
  - OLDMAC –Adds delimiter for old MACs (carriage return)
  - CRLF –Adds delimiter carriage return line feed (WIN)
  - WINDOWS –Adds delimiter for WINDOWS (carriage return line feed)
  - DLM –Adds the system specific delimiter (DEFAULT)
  - SYSTEM –Adds the system specific delimiter (DEFAULT)

4.2.121 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   CONV.WRITE.BINARY
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control character handling is a special feature for record oriented files of mainframe system. For print control, the ASA or machine control charaters are encoded in the first byte of the record.

If an attribute with such a control character is known on write operation, you can decide whether it will be part of the output (RETAIN), written to the output if required (DETACH) or never written (ERASE).

SELECTIONS

- DETACH –Manage print control character as record attribute
- RETAIN –Leave print control character as part of the data record
- ERASE –Erase the print control character from the record

4.2.122 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for write operation
PATH:   CONV.WRITE.BINARY
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.2.123 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of output data
PATH:   CONV.WRITE.BINARY
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- `NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]`

  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.2.124 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   CONV.WRITE.BINARY.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.2.125 OBJECT *CHARACTER*

SYNOPSIS

```
HELP:   Write character data to a file
PATH:   CONV.WRITE
TYPE:   OBJECT
SYNTAX: CHARACTER(FILE='str'/STREAM/DUMMY,FALLOC(),REMAIN,APPEND,FLUSH,METHOD=L4I/L4X/B4I/ ↩
    B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX,CCSID='str'/DEFAULT/ASCII/EBCDIC,CASE=UPPER/LOWER/ ↩
    FOLD/SUPPER/SLOWER/USRTAB,BOM,CHRMODE=STOP/IGNORE/SUBSTITUTE,ELF2NL,SUBCHAR[num/SYSTEM ↩
    ...],SYSTABLE=ICONV,USRTABLE='str'/NPAS/SEPA/DELA/DLAX,REPORTFILE='str'/STDOUT/STDERR, ↩
    COMPRESS/CMP.{},ENCRYPT/ENC.{},ENCODE/COD.{},FLAM4NDC,PRNCONTROL=DETACH/RETAIN/ERASE, ↩
    FRCREC,LANG='str',PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH())
```

DESCRIPTION Write character formats all UTF-8 text elements to one block. With this UTF-8 based text block a character set conversion is done. After this such block can then optionally compressed, encrypted and encoded and is written or appended to the specified file.

During the character conversion it supports:

- Stop, ignore or substitution of invalid or incomplete character
- A free defined substitution character list
- Different system substitution / transliteration tables
- A user defined substitution table to change the system tables
- Case management (upper, lower, folding and special casing)
- Transliteration between subsets (mapping) and normalization
- BOM management and automatic detection of char sets
- Reporting of ignored or substituted characters

For compression, encryption and encoding one of the supported methods can be used. For each of these methods specific parameters can be set to control the out bound conversions.

ARGUMENTS

- STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to write ["==origin.ext]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- SWITCH:REMAIN -Remain existing files (don't overwrite) [FALSE]
- SWITCH:APPEND -Append data to this file [FALSE]
- SWITCH:FLUSH -Enforce flush in front of close file [FALSE]
- NUMBER:METHOD=L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX -Method to convert records in blocks [NONE]

  - L4I -Adds 4 byte length fields:Little endian integer, length inclusive
  - L4X -Adds 4 byte length fields:Little endian integer, length exclusive (ZIP)
  - B4I -Adds 4 byte length fields:Big endian integer, length inclusive
  - B4X -Adds 4 byte length fields:Big endian integer, length exclusive (USS)
  - S4I -Adds 4 byte length fields:System endian integer, length inclusive
  - S4X -Adds 4 byte length fields:System endian integer, length exclusive (VAR)
  - HLI -Adds 4 byte length fields:Little endian short (LL00), length inclusive
  - HLX -Adds 4 byte length fields:Little endian short (LL00), length exclusive
  - HBI -Adds 4 byte length fields:Big endian short (LL00), length inclusive (MVS)
  - HBX -Adds 4 byte length fields:Big endian short (LL00), length exclusive
  - HSI -Adds 4 byte length fields:System endian short (LL00), length inclusive
  - HSX -Adds 4 byte length fields:System endian short (LL00), length exclusive

- SWITCH:FLAM4NDC -Write data as FLAM4FILE in MODE=NDC(No Data Compression) [FALSE]
- SWITCH:FRCREC -Enforce record orientation on record oriented devices [FALSE]

4.2.126 OBJECT *FALLOC*

SYNOPSIS

```
HELP:   Platform dependent file allocation parameter
PATH:   CONV.WRITE.CHARACTER
TYPE:   OBJECT
SYNTAX: FALLOC(ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZE=num,RECFORMAT=NONE/BIN/TXT/ ↩
    DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX- ↩
    REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR ↩
    /FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/ ↩
    VRA/VRM,RECLENGTH=num,KEYDISP=OLD/NEW/DEL,KEYPOSITION=num,KEYLENGTH=num,SUBSYSTEM(), ↩
    SPACE(),LIKE='str',RENEW)
```

DESCRIPTION The file allocation object collects several parameters for platforms which are record oriented based on a catalog system like z/OS® and where the space, volume and other things must be predefined. The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

But also on distributed platforms like (WIN, UNIX, MAC, ...) FLAM supports record oriented file types to keep the record information on such platforms. For this case the different record format for the open world can be defined. To read such a special file format on an open platform the correct record format must also be defined. On a catalog based system, this is not required. If the record format in the catalog are not the same as specified an error will occur.

ARGUMENTS

- NUMBER:ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [SEQ]
  - SEQ -Sequential data set [DEFAULT]
  - LIB -Library (ZOS-PDSE)
  - PDS -Partitioned data set (ZOS-PDS)
  - USS -UNIX files (path based)
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)
- NUMBER:BLKSIZE=num -Block size [system]

- NUMBER:RECFORMAT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format [auto]
  - NONE -No format defined
  - BIN -Open -Binary (undefined) data (only the last record are shorter then record length)
  - TXT -Open -Text data with standard delimiter (result is a stripped variable length records)
  - DLM -Open -Records separated by a special delimiter with the result of variable length records
  - VAR -Open -Variable length record (if necessary define length format)
  - VAR-ASA -Open -Variable length record with ASA print control character (if necessary define length format)
  - VAR-MCC -Open -Variable length record with machine print control codes (if necessary define length format)
  - FIX -Open -Fix length record (file size =N * record length)
  - FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
  - FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
  - VAR-REL -Open -Variable relative record (if necessary define length format)
  - VAR-REL-ASA -Open -Variable relative record with ASA print control character (if necessary define length format)
  - VAR-REL-MCC -Open -Variable relative record with machine print control codes (if necessary define length format)
  - FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
  - FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size =N * (record length + 9))
  - FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size =N * (record length + 9))
  - F -Host -Fixed-length, unblocked
  - FA -Host -Fixed-length, unblocked, ASA print control characters
  - FM -Host -Fixed-length, unblocked, machine print control codes
  - FS -Host -Fixed-length, standard
  - FSA -Host -Fixed-length, standard, ASA print control characters
  - FSM -Host -Fixed-length, standard, machine print control codes
  - FB -Host -Fixed-length, blocked
  - FBA -Host -Fixed-length, blocked, ASA print control characters
  - FBM -Host -Fixed-length, blocked, machine print control codes
  - FBS -Host -Fixed-length, blocked, standard
  - FBSA -Host -Fixed-length, blocked, standard, ASA print control characters
  - FBSM -Host -Fixed-length, blocked, standard, machine print control codes
  - FE -Host -Fixed-length, ESDS
  - FEA -Host -Fixed-length, ESDS, ASA print control characters
  - FEM -Host -Fixed-length, ESDS, machine print control codes

  - FK -Host -Fixed-length, KSDS
  - FKA -Host -Fixed-length, KSDS, ASA print control characters
  - FKM -Host -Fixed-length, KSDS, machine print control codes
  - FR -Host -Fixed-length, RRDS
  - FRA -Host -Fixed-length, RRDS, ASA print control characters
  - FRM -Host -Fixed-length, RRDS, machine print control codes
  - U -Host -Undefined-length
  - UA -Host -Undefined-length, ASA print control characters
  - UM -Host -Undefined-length, machine print control codes
  - V -Host -Variable
  - VA -Host -Variable, ASA print control characters
  - VM -Host -Variable, machine print control codes
  - VS -Host -Variable, spanned
  - VSA -Host -Variable, spanned, ASA print control characters
  - VSM -Host -Variable, spanned, machine print control codes
  - VB -Host -Variable, blocked
  - VBA -Host -Variable, blocked, ASA print control characters
  - VBM -Host -Variable, blocked, machine print control codes
  - VBS -Host -Variable, blocked, spanned
  - VBSA -Host -Variable, blocked, spanned, ASA print control characters
  - VBSM -Host -Variable, blocked, spanned, machine print control codes
  - VE -Host -Variable, ESDS
  - VEA -Host -Variable, ESDS, ASA print control characters
  - VEM -Host -Variable, ESDS, machine print control codes
  - VK -Host -Variable, KSDS
  - VKA -Host -Variable, KSDS, ASA print control characters
  - VKM -Host -Variable, KSDS, machine print control codes
  - VR -Host -Variable, RRDS
  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes
- NUMBER:RECLENGTH=num -Record length (data only, no print control and length fields)
- NUMBER:KEYDISP=OLD/NEW/DEL -Key disposition for index/key sequential data sets [OLD]
  - OLD -Key disposition OLD (keep the existing key)
  - NEW -Key disposition NEW (insert record number as key)
  - DEL -Key disposition DEL (delete key from record)
- NUMBER:KEYPOSITION=num -Key position (>=1) for index/key sequential data sets [1]
- NUMBER:KEYLENGTH=num -Key length (>=1) for index/key sequential data sets [8]
- STRING:LIKE='str' -Specifies the name of the model data set from which the attributes are to be copied
- SWITCH:RENEW -Enforce new allocation by a remove of an existing file

4.2.127 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   CONV.WRITE.CHARACTER.FALLOC
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.2.128 OBJECT *SPACE*

SYNOPSIS

```
HELP:   Platform dependent space definition parameter
PATH:   CONV.WRITE.CHARACTER.FALLOC
TYPE:   OBJECT
SYNTAX: SPACE(PRIMARY=num,SECONDARY=num,DIRBLOCKS=num,VOLSER='str',VOLUNIT='str',DATACLASS ↩
    ='str',MGMTCLASS='str',STORCLASS='str',NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG, ↩
    ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG,RELEASE=YES/NO,UNMOVABLE,CONTIG,PERMANENT,RLS= ↩
    NRI/CR/CRE)
```

DESCRIPTION The space definition object collects several parameters for platforms where data sets are preallocated like z/OS® and where clylinders or tracks must be predefined for a file allocation.

The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

All size values must be given in megabyte (MiB). The calculation in the platform dependent units (tracks, cylinders, ...) are done inernally. Note: 1 MiB = 1024 KiB = 1024*1024 Byte

ARGUMENTS

- `NUMBER:PRIMARY=num -Primary size in megabytes`
- `NUMBER:SECONDARY=num -Secondary size in megabytes`
- `NUMBER:DIRBLOCKS=num -Directory blocks for partitioned data sets`

- STRING:VOLSER='str' -Volume serial number
- STRING:VOLUNIT='str' -Volume unit
- STRING:DATACLASS='str' -Data class
- STRING:MGMTCLASS='str' -Management class
- STRING:STORCLASS='str' -Storage class
- NUMBER:NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for normal end [CATALOG]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for abnormal end [DELETE]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:RELEASE=YES/NO -Release unused space when file is closed [SEQ=YES, PDS=NO]
  - YES -Yes, do it
  - NO -No, don't do it
- SWITCH:UNMOVABLE -Allocate space unmovable [FALSE]
- SWITCH:CONTIG -Allocate space contiguously [FALSE]
- SWITCH:PERMANENT -Allocate space permanently [FALSE]
- NUMBER:RLS=NRI/CR/CRE -Specifies the type of record level sharing (RLS) being done for a specific data set [NONE]
  - NRI -No Read Integrity
  - CR -Consistent Read
  - CRE -Consistent Read Explicit

4.2.129 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion to this CCSID [DEFAULT]
PATH:   CONV.WRITE.CHARACTER
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC
```

DESCRIPTION The TO code defines the encoding string or CCSID for the output text stream. The default value of this CCSID for read operations is UTF-8. For write operations, the default value is derived from the environment variable LANG. If LANG is not set, the default ASCII or EBCDIC system-codepage is used, depending on the platform.

To get a list of supported encoding strings, please use the command below:

```
   flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit. Optionally, you can use one of the keywords below to use the system-specific default ASCII or EBCDIC CCSID.

SELECTIONS

- DEFAULT -Use default CCSID (system-specific)
- ASCII -Use default ASCII CCSID (environment)
- EBCDIC -Use default EBCDIC CCSID (environment)

4.2.130 PARAMETER *CASE*

SYNOPSIS

```
HELP:   Define upper, lower or special case conversion [NONE]
PATH:   CONV.WRITE.CHARACTER
TYPE:   NUMBER
SYNTAX: CASE=UPPER/LOWER/FOLD/SUPPER/SLOWER/USRTAB
```

DESCRIPTION With the case modifier, you can perform conversion to upper or lower case as well as apply case folding and some special case conversions.

Case folding converts all characters to lower case but is more compatible with language specific rules for doing so. For example, this may be useful for case-insensitive string comparison if the strings contain non-Latin characters (code point < 128).

This mapping is done for each code point in front of substitution or transliteration. The respective case is accomplished on the basis of a separate Unicode conversion table in memory. This conversion can be manipulated if user mapping is activated and mapping definitions ($\star$) are defined through the provided user table.

The options UPPER and LOWER convert characters without expansion, i.e. returned strings will never grow in length. Note, however, that case folding and the special options (convert e.g. ß to SS) can result in an expansion of data.

All these conversions are pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- UPPER -Change character to upper case
- LOWER -Change character to lower case
- FOLD -Change character to folded case (for compare)
- SUPPER -Change character to special upper case
- SLOWER -Change character to special lower case
- USRTAB -Change character according to user table

4.2.131 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Write byte order mark in front of the output text [FALSE]
PATH:   CONV.WRITE.CHARACTER
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the write byte order mark keyword is defined, then the corresponding BOM sign is written at the start of the output stream (block oriented) or at the beginning of each record. If this BOM keyword is not used, then the BOM sign will not be written (this is valid independent of the chosen encoding/CCSID). If a valid BOM sign is located in the middle of the input stream, it is removed from the data. If you define the BOM keyword and the target charset contains only single bytes or no valid BOM sign, then no byte order mark will be written to the output file.

If an invalid BOM sign is recognized, then it will be ignored and causes no entry in the report file. If the byte order changes in the middle of text and the BOM handling switch is not activated, the conversion will stop and an error message is thrown.

By default, the write BOM keyword is set to OFF.

4.2.132 PARAMETER *CHRMODE*

SYNOPSIS

```
HELP:   Define handling of non-convertible characters [STOP]
PATH:   CONV.WRITE.CHARACTER
TYPE:   NUMBER
SYNTAX: CHRMODE=STOP/IGNORE/SUBSTITUTE
```

DESCRIPTION With the MODE selection you can define how an invalid or incomplete character is handled inside of a unit by this component. All possible modes are listed at the end of this chapter.

This MODE is valid for the content of each unit (block or record). At the end of a unit there can be an incomplete character. Such a rest will be handled dependent on the METHOD you choose. The rest of the last unit of a data field will be concatenated to the first unit of the next data field. For all other records in a data field the handling of such a rest will be defined by the respective METHOD.

The SKIP case for METHOD is only relevant for incomplete byte sequences at the end of a unit. The ignore and substitution mode is only effective for all characters in the middle of the unit, besides an incomplete byte sequence occurs in the last unit. Please don't mix up the IGNORE case for MODE with the SKIP case of METHOD.

If a system or user substitution table is used the stop, ignore or substitute are only relevant for characters not managed by the CCSIDs and the substitution tables. This means that the USRTAB and SYSTAB are added to the capabilities of character conversion and only for the rest the chosen MODE is used.

4.2.132.1 CONSTANT *STOP*

SYNOPSIS

```
HELP:   Stop at the first non-convertible character
PATH:   CONV.WRITE.CHARACTER.CHRMODE
TYPE:   NUMBER
SYNTAX: STOP
```

DESCRIPTION Stops conversion at the first incomplete or invalid character which can not be converted. An error message is given with the position of abortion and the value in the input stream. An entry will be (STP()) written to the report file to log this event. To be compatible with other conversion tools, this is the default case, but we recommend to use IGNORE or SUBSTITUTE to prevent any abortion.

indexterm:[Constant STOP}

4.2.132.2 CONSTANT *IGNORE*

SYNOPSIS

```
HELP:   Ignore non-convertible characters
PATH:   CONV.WRITE.CHARACTER.CHRMODE
TYPE:   NUMBER
SYNTAX: IGNORE
```

DESCRIPTION Ignores incomplete or invalid characters and writes a report entry for each character. IGNORE is a substitution by no corresponding character that means no substitution table is required.

indexterm:[Constant IGNORE}

4.2.132.3 CONSTANT *SUBSTITUTE*

SYNOPSIS

```
HELP:   Substitute non-convertible characters
PATH:   CONV.WRITE.CHARACTER.CHRMODE
TYPE:   NUMBER
SYNTAX: SUBSTITUTE
```

DESCRIPTION Substitutes incomplete or invalid character and writes a report entry for each character. For SUBSTITUTE you can define a substitution character list (SUBCHR). If no SUBCHR is provided then this option substitutes invalid or incomplete characters with code point 0x00001A. If a SUBCHR is defined all invalid or incomplete characters will be substituted by the defined list of code points that can be up to 8. If you specify a SUBCHR the substitution mode will be enabled.

indexterm:[Constant SUBSTITUTE}

4.2.133 PARAMETER *ELF2NL*

SYNOPSIS

```
HELP:   Replace EBCDIC line feed (0x25) by new line (0x15) on output
PATH:   CONV.WRITE.CHARACTER
TYPE:   SWITCH
SYNTAX: ELF2NL
```

DESCRIPTION The replacement of EBCDIC line feed control characters (0x25) by new line characters (0x15) at conversion to EBCDIC can be activated on non EBCDIC (ASCII) systems or deactivated on EBCDIC systems by enabling this switch. This is for example required to prepare text files on open platforms for USS on z/OS. The usual delimiter for text in ASCII and UTF is 0x0A and will normally be translated to 0x25 in EBCDIC. On z/OS(USS), however, 0x15 (NL) is expected.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

To ensure a correctly delimited text record, only a line feed and not a carriage return (0x0D) line feed (0x0A) is used. If text formatting is used with method=NL, this will result in a single LF on distributed systems and the internal conversion will be activated by default to produce a valid text file for z/OS under USS on Windows, Unix and other open platforms.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.2.134 PARAMETER *SUBCHAR*

SYNOPSIS

```
HELP:   Default substitution character list (maximal 8 code points
PATH:   CONV.WRITE.CHARACTER
TYPE:   NUMBER
SYNTAX: SUBCHAR[num/SYSTEM...]
```

DESCRIPTION As substitution character (SUBCHR) you can define a list with up to 8 UNICODE points which are used for the substitution of invalid or incomplete characters if the corresponding code point is not defined in the substitution table (see SUBTAB). If you assign to it the keyword SYSTEM the value 0x1A is written as default substitution character.

If you activate substitution and no SUBCHR is defined, then unsupported characters will be replaced by CODEPOINT 0x00001A.

On the other hand SUBCHR will replace every undefined code point in the substitution table, what is the same behavior when using a USRTAB with corresponding SUBCHR values for every possible code point (000000-200000). If SUBCHR but no MODE is defined the substitution mode will be activated.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `SYSTEM -Use system specific substitution char (0x1A for ASCII/UTF, 0x3F for EBCDIC)`

4.2.135 PARAMETER *SYSTABLE*

SYNOPSIS

```
HELP:   Define a preloaded system substitution table
PATH:   CONV.WRITE.CHARACTER
TYPE:   NUMBER
SYNTAX: SYSTABLE=ICONV
```

DESCRIPTION With the system table (SYSTAB) you can load a substitution table with a predefined transliteration schema. Entries in the USRTAB can be used to override predefined transliteration entries or add missing transliteration code points. The options listed at the end of this chapter are available.

If a substitution table is used then the MODE option (STOP/IGNORE/SUBSTITUTE) is only relevant for characters that cannot be converted or substituted.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `ICONV -Preload with ICONV//TRANSLIT substitution table`

4.2.136 PARAMETER *USRTABLE*

SYNOPSIS

```
HELP:   Name of the file containing the user conversion table
PATH:   CONV.WRITE.CHARACTER
TYPE:   STRING
SYNTAX: USRTABLE='str'/NPAS/SEPA/DELA/DLAX
```

DESCRIPTION With this parameter, you can choose a predefined subset or standard manipulation for the system tables via keyword (see `flcl help "xcnv.inp .sav.fil.cnv.chr.usrtab"` for available keywords). Alternatively, you can use a file containing a user defined substitution table (USRTAB).

For the user table file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
USRTAB=NPAS                          ; use XOEV predefined subset
USRTAB=DELA                          ; German Latin single byte subset
USRTAB='~.MYUSRTAB(USRTAB1)'         ; PDS on z/OS
USRTAB='<SYSUID>.MYUSRT02'           ; PS dataset on z/OS
USRTAB='~/myusrtab3.txt'             ; Unix path name
USRTAB='<HOME>\usrtab\myusrtab4.txt' ; Windows path name
```

A user defined substitution table has the syntax listed below:

```
usr_tab_file         -> usr_definition_list
usr_definition_list  -> usr_definition usr_definition_list
                     |  EMPTY
usr_definition       -> '(' codepoint_definition ')'
codepoint_definition -> cp_activation
                     |  cp_deactivation
                     |  cp_transliteration
                     |  cp_standard_mapping
                     |  cp_usr_case_mapping
cp_activation        -> ['+'] CODEPOINT  [ '-' CODEPOINT ]
cp_deactivation      ->  '-'  CODEPOINT  [ '-' CODEPOINT ]
cp_transliteration   -> ['+'] CODEPOINT '=' code_point_list
cp_standard_mapping  ->  '*'  CODEPOINT '=' code_point_list
cp_usr_case_mapping  ->  '^'  CODEPOINT '=' code_point_list
code_point_list      -> CODEPOINT separator code_point_list
                     |  CODEPOINT
                     |  EMPTY
separator            -> '/'
                     |  ','
                     |  ';'
```

Note: In the description below "/" is used as code point separator.

The optional signs in front of a code point (CP) have the following meaning:

```
'+' - a valid transliteration (only if CP not in the target set)
'*' - a valid mapping         (this translation is always done)
'^' - a case mapping/folding  (only if case=usrtab is activated)
'-' - an invalid definition   (removes CP for subset definitions)
```

If no sign is used, then a transliteration for a valid code point is defined. Valid code points are accepted, invalid code points result in in an error. Error handling depends on the mode (STOP,IGNORE, SUBSTITUTE). Definition of + code points without code point list, thus without = is equal definition of valid code points. With an asterisk $*$, you can define an enforced mapping for this code point. For example, this can be used to delete this character if no code point list is provided or to translate this character always in another value. This can also be used to convert code points outside of a subset into this subset.

In contrast to the transliteration (+ is only done if a character doesn't exist in the target encoding), the mapping is always done for each character. Invalid (−) code points can be used to deactivate a code point. To activate (+) or deactivate (−) a range of code points you can use the minus mark (−) followed by one sub code point. The range goes from the smaller code point to the bigger one. For example:

```
(-00-7F) deactivate all US-ASCII code points
(+39-30) activate all decimal digits
```

The range operator (−) with the optional plus sign (+) can also be used to activate code points.

To define an own case mapping or folding (required for certain subsets) the caret sign (^) can be used. This mapping is only done if the user table is activated for case mapping (CASE=USRTAB).

To define transliterations/substitutions/mappings an assignment (=) of a code point list is required. The code point list can contain a maximum of 8 code points. If no code point list is specified, then the character is ignored. A transliteration to itself simply activates this code point. The same is valid for mapping definitions. A code point is a hexadecimal number representing a 21 bit UNICODE point.

```
  00000000 to 001FFFFF hex
```

If a code point is not defined in the substitution table (USRTAB and/or SYSTAB), the appropriate substitution character (SUBCHR) is used. If SUBCHR is not set, no substitution is performed. Depending on the MODE, character conversion stops with an error (STOP) or ignores the character (IGNORE).

Text before and after brackets are comments.

Between "(" and "=" or "/" or ")" you can define hex digits until the first non-hex digit. All non-hex digits up to the next separator are interpreted as comment. Leading whitespace is ignored.

```
  REPLACE GERMAN SZ (00DF=0073/0073)                  WITH ss
THIS IS AN EXAMPLE FOR COMMENTS IN CODE POINT LIST:
  REPLACE EURO MARK (20AC= 45#E / 55#U / 52#R / 4F#O) WITH EURO
  REPLACE BOM MARK  (EFFF=)                           WITH NOTHING
```

Leading zeros are allowed. If you don't supply a hex value, then 0x00 is used.

```
  REPLACE GERMAN SZ (00DF=/)                   WITH 0x00 0x00
  REPLACE EURO MARK (20AC=00000045/00055/000052/000004F) WITH EURO
```

ATTENTION: Please don't use parentheses "()" or the operators in your comments.

To describe your own subsets, you can use a user table without a system table. The USRTAB can also be used to overwrite or add transliterations when a system substitution table (for example SYSTAB=ICONV) is used. The transliteration works recursively, that is if one of the substitution code points is not in the target set, this substitution will be used instead and so on.

```
  REPLACE GERMAN OE (0000D6=004F/0045)       WITH O E
  REPLACE GERMAN SZ (0000DF=0073/0073)       WITH s s
  REPLACE EURO MARK (0020AC=00D6/00DF/52/4F) WITH OE SZ R O
```

If you have a EURO sign in your text and convert it to *Latin1*, the resulting byte string will be *D6DF524F*. On the other hand, if you convert it to *ASCII*, the byte string will be *4F457373524F*.

The mapping as described above is always done and is also done recursive including the transliteration result. For example, if you define a mapping `(*30=39)` then the target data won't have any zero in the resulting text.

By replacing other code points recursively, you could easily cause infinite loops. To prevent this, the amount of replacements and the length of one replacement is limited to a maximum of 64. Note that this could still result in a large expansion of data. Therefore, be careful about defining recursive substitutions.

A sample user table for the ICONV system table to change the transliteration of German umlauts to AE, OE or UE is located in the SAMPLE directory under CCUTDEXL. Another sample user table called CCUTNPAS defines the *string.latin* subset (XOEF) which is mainly used for statutory reporting.

The order of definition is the order of processing the definitions. For exampel, if you define a mapping or a transliteration for code point X and later you make this code point invalid, then the code point is invalid. On the other hand, you can first deactivate all code points, then activate your subset and define your transliterations (best fit) or mappings.

For example: To define a non-expansion best fit mapping a user table can be used to delete all combined characters for single byte code pages. For the ICONV system transliteration table, a sample user table can be found under the name CCUTBFM1.

Some other subset user table definitions were added. For example:

- currently valid SEPA characters (<128) for money transfers (CCUTSEPA)
- characters valid in ISO-8859-15, CP1252 and IBM1142 (CCUTDELA)
- characters valid in ISO-8859-15, CP1252, IBM1142 and XOEF (CCUTDLAX)

All user table definitions are pre-calculated once at the beginning of execution to reduce the required CPU time for the conversion. So, the use of a user table increases the cost to open a file but has no effect on CPU utilization when converting the data.

To minimize the effort and to simplify the usage of user table, the most common subsets can be selected by keyword. In this case, a pre-calculated load module (DLL) is used. The subset definition can be for UTF (21 Bit UNICODE) or UCS (16 Bit UNICODE). If it is a UCS subset, then no valid code point is greater than 0xFFFF and you must use UCS CCSIDS to ensure that no code point greater than 0xFFFF is accepted.

If you have a working user table definition for a popular subset or system table manipulation, you can request a keyword for it. Please report this requirement over our issue tracking system

```
http://www.flam.de/en/technology/support/issues/
```

and attach the corresponding user table definition.

SELECTIONS

- `NPAS -UCS subset for statutory reporting (New passport, XOEF, String.Latin)`
- `SEPA -UCS subset of valid SEPA character smaller then 128`
- `DELA -UCS subset for German Latin of IBM1141, ISO8859-15 and CP1252`
- `DLAX -UCS subset for German Latin of IBM1141, ISO8859-15, CP1252 and XOEF`

4.2.137 PARAMETER *REPORTFILE*

SYNOPSIS

```
HELP:   Name of the report file to log substitutions
PATH:   CONV.WRITE.CHARACTER
TYPE:   STRING
SYNTAX: REPORTFILE='str'/STDOUT/STDERR
```

DESCRIPTION With the report capability, you can log every substitution (stop / ignore / substitute / translit / skip / ...) to a log file. For the report file, all rules and replacements are valid, which are described under "FILENAME HANDLING" above. For example:

```
REPORT=STDERR                           ; is SYSPRINT on z/OS
REPORT='~.MYREPORT(REPORT1)'            ; PDS on z/OS
REPORT='<SYSUID>.MYREPRT2'              ; PS dataset on z/OS
REPORT='~/myreport3.txt'                ; Unix path name
REPORT='<HOME>\reports\myreport4.txt'   ; Windows path name
```

A substitution report has the syntax listed below:

```
report_file         -> sub_definition_list
sub_definition_list -> sub_definition sub_definition_list
                    |  EMPTY
sub_definition      -> SUB '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    -> XLT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    |  IGN '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  STP '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  IDT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  SKP '(' UNIT ':' OFFSET '+' POSITION ':'
```

```
                          code_point byte_string ')'
                    |  UKW '(' UNIT ':' OFFSET '+' POSITION ':'
                          code_point byte_string ')'
  code_point_list   -> CODEPOINT '/' code_point_list
                    |  EMPTY
  code_point        -> CODEPOINT
                    |  EMPTY
  byte_string       -> '(' BYTESTRING ')'
                    |  EMPTY
```

A comment behind brackets explains the reason for substitution:

```
'invalid character'
'incomplete character'
'incomplete character at the end of a record'
```

The notations STP (stop), IGN (ignore), SUB (substitute), XLT (transliteration), IDT (identical) or SKP (skip) before round brackets indicate the action taken. If the task can not be determined then UKW (unknown) is used. A ignore of an invalid character is realized as substitution with nothing. All substitutions with a length of 0 are indicated with IGN (ignore). The position of such an abnormal/irregular character is coded inside the braces:

- UNIT number of records or blocks (depends on the data)
- OFFSET byte offset up to this unit in the complete data stream
- POSITION number of bytes inside the unit where the activity was done

All counting starts with 0. The first block or record has unit 0.

The position is relative to the data stream which was presented to the character conversion module. After conversions and formatting the position values can be far off from the input file which was read originally. For example, when reading a text file with text formatting to records, then delimiting characters are not included in the byte offset, the units are records and the position indicates which character was the issue for this log entry.

The CODEPOINTs (UNICODE) are taken from the defined substitution table The BYTESTRINGs are taken from the input (read characters) and output data (written characters). BYTESTRINGs and CODEPOINTs are provided if they are known in the cases of such an irregularity. In some cases, these values can be missing.

For interpreting the report file, you must know that only the values inside round brackets are meaningful. Text before or behind round brackets are comments, but this may be subject to changes.

SELECTIONS

- `STDOUT -Write output to stdout`
- `STDERR -Write output to stderr`

4.2.138 OVERLAY *COMPRESS*

SYNOPSIS

```
HELP:   Compression procedure [NONE]
PATH:   CONV.WRITE.CHARACTER
TYPE:   OVERLAY
SYNTAX: COMPRESS/CMP.{GZIP()/BZIP2()/XZ()/FLAM4()}
```

DESCRIPTION With the overlay compress you can choose which kind of compression method is used for the data. Depending on the compression method different parameter can be provided to define the behavior in more detail.

If no compression method selected no compression of the data will be done.

See below the different possibilities of compression.

4.2.139 OBJECT *GZIP*

SYNOPSIS

```
HELP:   Compress data compliant to RCF1952 (GZIP)
PATH:   CONV.WRITE.CHARACTER.COMPRESS
TYPE:   OBJECT
SYNTAX: GZIP(LEVEL=num/COPY/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO, ↩
    COMMENT='str',CHECK)
```

DESCRIPTION The GZIP compression conforms to RCF1951/52. The compression level can be set to values from 0 to 9 (0=copy, 1=fastest, 9=best). Additionally, it is possible to set a comment and to enforce checksum calculation of the file header.

The checksum calculation is not recommended because only the newer versions of Zlib can handle this checksum technique. With older versions of Zlib this can cause problems. Therefore, checksum calculation is turned off by default.

The decompressor also accepts the older Zlib wrapping format specified in RFC1950. It is not possible to enforce RFC1950 at compression because most standards require RFC1952 for the deflate algorithms.

If no compression level is defined and the data is marked is being without redundancy, level 0 (copy) is used. In all other cases the default level 6 is used.

Decompression of concatenated GZIP files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

`filename/:memberindex`

ARGUMENTS

- `NUMBER:LEVEL=num/COPY/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level [AUTO==6]`
  - `COPY -No compression =copy of data (0)`
  - `FAST -Fastest compression (level 1)`
  - `LEV1 -Compression level 1`
  - `LEV2 -Compression level 2`
  - `LEV3 -Compression level 3`
  - `LEV4 -Compression level 4`
  - `LEV5 -Compression level 5`
  - `LEV6 -Compression level 6`
  - `LEV7 -Compression level 7`
  - `LEV8 -Compression level 8`
  - `LEV9 -Compression level 9`
  - `BEST -Best compression (level 9)`
  - `AUTO -Automatic compression (level 6)`
- `STRING:COMMENT='str' -Comment for header [NONE]`
- `SWITCH:CHECK -Calculate checksum over GZIP header [FALSE]`

4.2.140 OBJECT *BZIP2*

SYNOPSIS

```
HELP:   BZIP2 data compression technique
PATH:   CONV.WRITE.CHARACTER.COMPRESS
TYPE:   OBJECT
SYNTAX: BZIP2(LEVEL=num/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO)
```

DESCRIPTION The BZIP2 algorithm compresses chunks of data. Such a chunk can be up to 100KB, 200KB, . . . , 900KB. The chunk size is selected automatically based on the internal block size (block size in KB rounded up to the next possible chunk size).

The chunk size directly correlates with the compression level. Compression levels range from 1 (fastest) to 9 (best, but slowest). Level 1 results in 100KB chunk size, level 9 in 900KB chunk size (i.e. chunk size = level*100KB). Hence, a lower level also reduces the required memory resources during compression and decompression.

Decompression of concatenated BZIP2 files is also possible. All concatenated files will be decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
```

ARGUMENTS

- `NUMBER:LEVEL=num/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level (defines block size) [AUTO]`
  - `FAST -Fastest compression (Block size =100kb)`
  - `LEV1 -Compression level 1 (Block size =100kb)`
  - `LEV2 -Compression level 2 (Block size =200kb)`
  - `LEV3 -Compression level 3 (Block size =300kb)`
  - `LEV4 -Compression level 4 (Block size =400kb)`
  - `LEV5 -Compression level 5 (Block size =500kb)`
  - `LEV6 -Compression level 6 (Block size =600kb)`
  - `LEV7 -Compression level 7 (Block size =700kb)`
  - `LEV8 -Compression level 8 (Block size =800kb)`
  - `LEV9 -Compression level 9 (Block size =900kb)`
  - `BEST -Best compression (Block size =900kb)`
  - `AUTO -Automatic compression (depending on the real block size)`

4.2.141 OBJECT *XZ*

SYNOPSIS

```
HELP:   XZ (LZMA) data compression technique
PATH:   CONV.WRITE.CHARACTER.COMPRESS
TYPE:   OBJECT
SYNTAX: XZ(LEVEL=num/FAST/LEV0/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO)
```

DESCRIPTION The XZ compression conforms to the XZ-compression tool on UNIX-systems. The compression level can be set to values from 0 to 9 (0=fastest, 9=best).

The default level is 6 when no level is specified.

Decompression of concatenated XZ files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
```

ARGUMENTS

- `NUMBER:LEVEL=num/FAST/LEV0/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level [AUTO==6]`
  - `FAST -Fastest compression (level 0)`

- LEV0 -Compression level 0
- LEV1 -Compression level 1
- LEV2 -Compression level 2
- LEV3 -Compression level 3
- LEV4 -Compression level 4
- LEV5 -Compression level 5
- LEV6 -Compression level 6
- LEV7 -Compression level 7
- LEV8 -Compression level 8
- LEV9 -Compression level 9
- BEST -Best compression (level 9)
- AUTO -Automatic compression (level 6)

4.2.142 OBJECT *FLAM4*

SYNOPSIS

```
HELP:   Compress data with FLAM
PATH:   CONV.WRITE.CHARACTER.COMPRESS
TYPE:   OBJECT
SYNTAX: FLAM4(MODE=CX7/CX8/VR8/ADC/NDC,COMMENT='str')
```

DESCRIPTION The FLAM4 compression to write compressed data into FLAMFILEs. As main argument you can define the compression mode (ADC, VR8, CX8). Additionally, it is possible to set a comment in the file header.

If the record-oriented method (write.record()) is used, the records with all attributes are written to the FLAM archive. If a block-oriented method (write.binary/char/text/xml()) is used, the data blocks are wrapped into records and written to the FLAMFILE as binary chunks with an undefined record format (U).

ARGUMENTS

- NUMBER:MODE=CX7/CX8/VR8/ADC/NDC -Compression mode [ADC]
  - CX7 -Compression encoding as printable characters
  - CX8 -Compression encoding as binary data
  - VR8 -Enhanced compression encoding as binary data
  - ADC -Advanced Data Compression
  - NDC -No data compression
- STRING:COMMENT='str' -Comment for the member in the FLAMFILE [internal use]

4.2.143 OVERLAY *ENCRYPT*

SYNOPSIS

```
HELP:   Encryption procedure [NONE]
PATH:   CONV.WRITE.CHARACTER
TYPE:   OVERLAY
SYNTAX: ENCRYPT/ENC.{F4PWD()/F4KME()}
```

DESCRIPTION The overlay *encrypt* allows you to choose which kind of encryption method is used for the data. Depending on the encryption method, different parameters can be set.

If no encryption method is selected, then no data encryption will be done.

See below for the encryption possibilities.

4.2.144 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password encryption
PATH:   CONV.WRITE.CHARACTER.ENCRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(MODE=AES/FLAM,PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Activates FLAM4 password-based encryption to encrypt the data written to FLAMFILEs. You can choose the encryption mode. The default and recommended mode is AES. If you provide no password, a hardcoded default password is used.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

If the data is already compressed, then MODE=NDC (no data compression) is used. Otherwise, ADC compression is used. If you set the compression mode to CX7/CX8/VR8 and enable encryption, e.g.

```
  comp.flam(mode=CX7/CX8/VR8) encr.f4pwd()
```

the compression mode is ignored and ADC/NDC is used instead. Due to how those three old compression modes work, they are incompatible with FLAM encryption.

ARGUMENTS

- `NUMBER:MODE=AES/FLAM -Encryption mode [AES]`
  - `AES -AES encryption`
  - `FLAM -FLAM encryption`
- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['******']`
  - `DEFAULT -FLAM4 default password`

4.2.145 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   CONV.WRITE.CHARACTER.ENCRYPT
TYPE:   OBJECT
SYNTAX: F4KME(MODE=AES/FLAM,LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 encryption of data written to FLAMFILEs based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME parameter is provided (encr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using encr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

You may define the encryption mode. The default and recommended mode is AES.

If the data is already compressed, then MODE=NDC (no data compression) is used. Otherwise, ADC compression is used to create the FLAMFILE. If you set the compression mode to CX7/CX8/VR8 and enable encryption, e.g.

```
comp.flam(mode=CX7/CX8/VR8) encr.f4kme()
```

the compression mode is ignored and ADC/NDC is used instead. Due to how those three old compression modes work, they are incompatible with FLAM encryption.

ARGUMENTS

- `NUMBER:MODE=AES/FLAM -Encryption mode [AES]`
  - `AES -AES encryption`
  - `FLAM -FLAM encryption`
- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME [”]`

4.2.146 OVERLAY *ENCODE*

SYNOPSIS

```
HELP:   Encoding procedure [NONE]
PATH:   CONV.WRITE.CHARACTER
TYPE:   OVERLAY
SYNTAX: ENCODE/COD.{BASE64()/BASE32()/BASE16()/OPGP()}
```

DESCRIPTION The overlay *encode* allows to choose which kind of encoding method is used for the data. Depending on the encoding method, different parameters can be set.

If no encoding method is selected, then no data encoding will be done.

See below for the available encoding options.

4.2.147 OBJECT *BASE64*

SYNOPSIS

```
HELP:   Base64 encoding (RFC4648)
PATH:   CONV.WRITE.CHARACTER.ENCODE
TYPE:   OBJECT
SYNTAX: BASE64(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)

  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.148 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.CHARACTER.ENCODE.BASE64
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.149 OBJECT *BASE32*

SYNOPSIS

```
HELP:   Base32 encoding (RFC4648)
PATH:   CONV.WRITE.CHARACTER.ENCODE
TYPE:   OBJECT
SYNTAX: BASE32(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.150 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.CHARACTER.ENCODE.BASE32
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
```

```
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.151 OBJECT *BASE16*

SYNOPSIS

```
HELP:   Base16 encoding (RFC4648)
PATH:   CONV.WRITE.CHARACTER.ENCODE
TYPE:   OBJECT
SYNTAX: BASE16(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)

  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)

- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.152 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.CHARACTER.ENCODE.BASE16
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.153 OBJECT *OPGP*

SYNOPSIS

```
HELP:   Base64 encoding with 76 character, CRC checksum and Armor clasp (RFC4880)
PATH:   CONV.WRITE.CHARACTER.ENCODE
TYPE:   OBJECT
SYNTAX: OPGP(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE ↩
    /UTF32BE/UCS4LE/UTF32LE,DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM)
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- `NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]`
  - `NONE -No character set defined`
  - `SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)`
  - `ASCII -ASCII (mainly used in the for open system)`
  - `UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)`
  - `UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)`
  - `EBCDIC -EBCDIC (mainly used on IBM mainframe)`
  - `UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)`
  - `UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)`
  - `UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)`
  - `UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)`
  - `UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)`

  - UTF32BE –UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE –UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE –UTF-32 Little Endian (multibyte character set < 2M)

- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM –Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST –Adds no delimiter (HOST)
  - BIN –Adds no delimiter (HOST)
  - NL –Adds delimiter new line (USS)
  - USS –Adds delimiter for USS (new line)
  - LF –Adds delimiter line feed (UNIX)
  - UNIX –Adds delimiter for UNIX (line feed)
  - CR –Adds delimiter carriage return (MAC)
  - OLDMAC –Adds delimiter for old MACs (carriage return)
  - CRLF –Adds delimiter carriage return line feed (WIN)
  - WINDOWS –Adds delimiter for WINDOWS (carriage return line feed)
  - DLM –Adds the system specific delimiter (DEFAULT)
  - SYSTEM –Adds the system specific delimiter (DEFAULT)

4.2.154 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   CONV.WRITE.CHARACTER
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control character handling is a special feature for record oriented files of mainframe system. For print control, the ASA or machine control charaters are encoded in the first byte of the record.

If an attribute with such a control character is known on write operation, you can decide whether it will be part of the output (RETAIN), written to the output if required (DETACH) or never written (ERASE).

SELECTIONS

- DETACH –Manage print control character as record attribute
- RETAIN –Leave print control character as part of the data record
- ERASE –Erase the print control character from the record

4.2.155 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for write operation
PATH:   CONV.WRITE.CHARACTER
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.2.156 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for write operation
PATH:   CONV.WRITE.CHARACTER
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.2.157 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of output data
PATH:   CONV.WRITE.CHARACTER
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
OUTPUT=STDERR                          ; is SYSPRINT on z/OS
OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
OUTPUT='~/mycheck3.txt'                ; Unix path name
OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.2.158 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   CONV.WRITE.CHARACTER.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- `STRING:VALUE='str' -Checksum as hex-string`
- `STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`

4.2.159 OBJECT *TEXT*

SYNOPSIS

```
HELP:   Write text data to a file
PATH:   CONV.WRITE
TYPE:   OBJECT
SYNTAX: TEXT(FILE='str'/STREAM/DUMMY,FALLOC(),REMAIN,APPEND,FLUSH,METHOD=HOST/BIN/REC/NL/ ↩
    USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM/ORIGINAL,RPLTAB/RPLHTB=num,RPLVTB=num, ↩
    RPLCTR=SPACE/SUBSTITUTE/DELETE,SUPCTR,SUPTWS,CCSID='str'/DEFAULT/ASCII/EBCDIC,CASE=UPPER ↩
    /LOWER/FOLD/SUPPER/SLOWER/USRTAB,BOM,ELF2NL,CHRMODE=STOP/IGNORE/SUBSTITUTE,SUBCHAR[num/ ↩
    SYSTEM...],SYSTABLE=ICONV,USRTABLE='str'/NPAS/SEPA/DELA/DLAX,REPORTFILE='str'/STDOUT/ ↩
    STDERR,COMPRESS/CMP.{},ENCRYPT/ENC.{},ENCODE/COD.{},FLAM4NDC,PRNCONTROL=DETACH/RETAIN/ ↩
    ERASE,FRCREC,LANG='str',PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,DUMP,HASH(),INDSIZ=num, ↩
    INDCHR=SPACE/TABULATOR,NOCMNT)
```

DESCRIPTION Write text formats UTF-8 text elements into a list of records / lines, each terminated by a delimiter. These text records are converted to a character set defined by the CSSID. The resulting record list can optionally be compressed, encrypted and/or encoded and is written or appended to the specified file.

Text formatting also supports:

- Different methods to define the record/line delimiter

- Suppression of trailing whitespace
- Replacement of horizontal tabs by spaces
- Suppression or replacement of remaining control characters

During character conversion these options are supported:

- Stopping, ignoring or substitution when encountering an invalid or incomplete character
- A freely defined substitution character list
- Different system substitution / transliteration tables
- A user-defined substitution table to change the system tables
- Case management (upper, lower, folding and special casing)
- Transliteration between subsets (mapping) and normalization
- BOM management and automatic detection of character sets
- Reporting of ignored or substituted characters

For compression, encryption and encoding one of the supported methods can be used. For each of these methods, specific parameters can be set to control the conversions.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to write ["==origin.ext]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`
- `SWITCH:REMAIN -Remain existing files (don't overwrite) [FALSE]`
- `SWITCH:APPEND -Append data to this file [FALSE]`
- `SWITCH:FLUSH -Enforce flush in front of close file [FALSE]`
- `NUMBER:METHOD=HOST/BIN/REC/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM/ORIGINAL -Method for text formatting [SYSTEM]`
  - `HOST -Adds no delimiter (HOST)`
  - `BIN -Adds no delimiter (HOST)`
  - `REC -Adds no delimiter (HOST)`
  - `NL -Adds delimiter new line (USS)`
  - `USS -Adds delimiter for USS (new line)`
  - `LF -Adds delimiter line feed (UNIX)`
  - `UNIX -Adds delimiter for UNIX (line feed)`
  - `CR -Adds delimiter carriage return (MAC)`
  - `OLDMAC -Adds delimiter for old MACs (carriage return)`
  - `CRLF -Adds delimiter carriage return line feed (WIN)`
  - `WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)`
  - `DLM -Adds the system specific delimiter (DEFAULT)`
  - `SYSTEM -Adds the system specific delimiter (DEFAULT)`
  - `ORIGINAL -Adds the original data at the end of a line (only for band FMT)`

- NUMBER:RPLTAB/RPLHTB=num -Replace horizontal tabulators by spaces using this tab width [0 -no replacement]
- NUMBER:RPLVTB=num -Replace vertical tabulators by new lines using this tab width [0 = no replacement]
- NUMBER:RPLCTR=SPACE/SUBSTITUTE/DELETE -Replace remaining control characters [NONE]
  - SPACE -Replace control characters with whitespace character (0x20/0x40)
  - SUBSTITUTE -Replace control characters with substitution character (0x1A/0x3F)
  - DELETE -Remove control characters
- SWITCH:SUPCTR -Suppress control characters [FALSE] (RPLCTR=DELETE)
- SWITCH:SUPTWS -Suppress trailing whitespaces [FALSE]
- SWITCH:FLAM4NDC -Write data as FLAM4FILE in MODE=NDC(No Data Compression) [FALSE]
- SWITCH:FRCREC -Enforce record orientation on record oriented devices [FALSE]
- SWITCH:DUMP -Write text dump of binary data [FALSE]
- NUMBER:INDSIZ=num -Number of XML indentation characters per indentation level [2 (SPACE) or 1 (TAB)]
- NUMBER:INDCHR=SPACE/TABULATOR -Indentation character used to indent XML tags [SPACE]
  - SPACE -Use the space character as indentation character
  - TABULATOR -Use the tabulator character as indentation character
- SWITCH:NOCMNT -Do not write out XML comments [FALSE]

4.2.160 OBJECT *FALLOC*

SYNOPSIS

```
HELP:   Platform dependent file allocation parameter
PATH:   CONV.WRITE.TEXT
TYPE:   OBJECT
SYNTAX: FALLOC(ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZE=num,RECFORMAT=NONE/BIN/TXT/ ↩
    DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX- ↩
    REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR ↩
    /FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/ ↩
    VRA/VRM,RECLENGTH=num,KEYDISP=OLD/NEW/DEL,KEYPOSITION=num,KEYLENGTH=num,SUBSYSTEM(), ↩
    SPACE(),LIKE='str',RENEW)
```

DESCRIPTION The file allocation object collects several parameters for platforms which are record oriented based on a catalog system like z/OS® and where the space, volume and other things must be predefined. The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

But also on distributed platforms like (WIN, UNIX, MAC, ...) FLAM supports record oriented file types to keep the record information on such platforms. For this case the different record format for the open world can be defined. To read such a special file format on an open platform the correct record format must also be defined. On a catalog based system, this is not required. If the record format in the catalog are not the same as specified an error will occur.

ARGUMENTS

- NUMBER:ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [SEQ]
  - SEQ -Sequential data set [DEFAULT]

  - LIB -Library (ZOS-PDSE)
  - PDS -Partitioned data set (ZOS-PDS)
  - USS -UNIX files (path based)
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)
- NUMBER:BLKSIZE=num -Block size [system]
- NUMBER:RECFORMAT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format [auto]
  - NONE -No format defined
  - BIN -Open -Binary (undefined) data (only the last record are shorter then record length)
  - TXT -Open -Text data with standard delimiter (result is a stripped variable length records)
  - DLM -Open -Records separated by a special delimiter with the result of variable length records
  - VAR -Open -Variable length record (if necessary define length format)
  - VAR-ASA -Open -Variable length record with ASA print control character (if necessary define length format)
  - VAR-MCC -Open -Variable length record with machine print control codes (if necessary define length format)
  - FIX -Open -Fix length record (file size =N * record length)
  - FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
  - FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
  - VAR-REL -Open -Variable relative record (if necessary define length format)
  - VAR-REL-ASA -Open -Variable relative record with ASA print control character (if necessary define length format)
  - VAR-REL-MCC -Open -Variable relative record with machine print control codes (if necessary define length format)
  - FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
  - FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size =N * (record length + 9))
  - FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size =N * (record length + 9))
  - F -Host -Fixed-length, unblocked
  - FA -Host -Fixed-length, unblocked, ASA print control characters
  - FM -Host -Fixed-length, unblocked, machine print control codes
  - FS -Host -Fixed-length, standard
  - FSA -Host -Fixed-length, standard, ASA print control characters
  - FSM -Host -Fixed-length, standard, machine print control codes
  - FB -Host -Fixed-length, blocked
  - FBA -Host -Fixed-length, blocked, ASA print control characters

  - FBM -Host -Fixed-length, blocked, machine print control codes
  - FBS -Host -Fixed-length, blocked, standard
  - FBSA -Host -Fixed-length, blocked, standard, ASA print control characters
  - FBSM -Host -Fixed-length, blocked, standard, machine print control codes
  - FE -Host -Fixed-length, ESDS
  - FEA -Host -Fixed-length, ESDS, ASA print control characters
  - FEM -Host -Fixed-length, ESDS, machine print control codes
  - FK -Host -Fixed-length, KSDS
  - FKA -Host -Fixed-length, KSDS, ASA print control characters
  - FKM -Host -Fixed-length, KSDS, machine print control codes
  - FR -Host -Fixed-length, RRDS
  - FRA -Host -Fixed-length, RRDS, ASA print control characters
  - FRM -Host -Fixed-length, RRDS, machine print control codes
  - U -Host -Undefined-length
  - UA -Host -Undefined-length, ASA print control characters
  - UM -Host -Undefined-length, machine print control codes
  - V -Host -Variable
  - VA -Host -Variable, ASA print control characters
  - VM -Host -Variable, machine print control codes
  - VS -Host -Variable, spanned
  - VSA -Host -Variable, spanned, ASA print control characters
  - VSM -Host -Variable, spanned, machine print control codes
  - VB -Host -Variable, blocked
  - VBA -Host -Variable, blocked, ASA print control characters
  - VBM -Host -Variable, blocked, machine print control codes
  - VBS -Host -Variable, blocked, spanned
  - VBSA -Host -Variable, blocked, spanned, ASA print control characters
  - VBSM -Host -Variable, blocked, spanned, machine print control codes
  - VE -Host -Variable, ESDS
  - VEA -Host -Variable, ESDS, ASA print control characters
  - VEM -Host -Variable, ESDS, machine print control codes
  - VK -Host -Variable, KSDS
  - VKA -Host -Variable, KSDS, ASA print control characters
  - VKM -Host -Variable, KSDS, machine print control codes
  - VR -Host -Variable, RRDS
  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes
- NUMBER:RECLENGTH=num -Record length (data only, no print control and length fields)
- NUMBER:KEYDISP=OLD/NEW/DEL -Key disposition for index/key sequential data sets [OLD]
  - OLD -Key disposition OLD (keep the existing key)
  - NEW -Key disposition NEW (insert record number as key)
  - DEL -Key disposition DEL (delete key from record)
- NUMBER:KEYPOSITION=num -Key position (>=1) for index/key sequential data sets [1]
- NUMBER:KEYLENGTH=num -Key length (>=1) for index/key sequential data sets [8]
- STRING:LIKE='str' -Specifies the name of the model data set from which the attributes are to be copied
- SWITCH:RENEW -Enforce new allocation by a remove of an existing file

4.2.161 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   CONV.WRITE.TEXT.FALLOC
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.2.162 OBJECT *SPACE*

SYNOPSIS

```
HELP:   Platform dependent space definition parameter
PATH:   CONV.WRITE.TEXT.FALLOC
TYPE:   OBJECT
SYNTAX: SPACE(PRIMARY=num,SECONDARY=num,DIRBLOCKS=num,VOLSER='str',VOLUNIT='str',DATACLASS ↩
    ='str',MGMTCLASS='str',STORCLASS='str',NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG, ↩
    ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG,RELEASE=YES/NO,UNMOVABLE,CONTIG,PERMANENT,RLS= ↩
    NRI/CR/CRE)
```

DESCRIPTION The space definition object collects several parameters for platforms where data sets are preallocated like z/OS® and where clylinders or tracks must be predefined for a file allocation.

The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

All size values must be given in megabyte (MiB). The calculation in the platform dependent units (tracks, cylinders, ...) are done inernally. Note: 1 MiB = 1024 KiB = 1024*1024 Byte

ARGUMENTS

- `NUMBER:PRIMARY=num -Primary size in megabytes`
- `NUMBER:SECONDARY=num -Secondary size in megabytes`
- `NUMBER:DIRBLOCKS=num -Directory blocks for partitioned data sets`

- STRING:VOLSER='str' -Volume serial number
- STRING:VOLUNIT='str' -Volume unit
- STRING:DATACLASS='str' -Data class
- STRING:MGMTCLASS='str' -Management class
- STRING:STORCLASS='str' -Storage class
- NUMBER:NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for normal end [CATALOG]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for abnormal end [DELETE]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:RELEASE=YES/NO -Release unused space when file is closed [SEQ=YES, PDS=NO]
  - YES -Yes, do it
  - NO -No, don't do it
- SWITCH:UNMOVABLE -Allocate space unmovable [FALSE]
- SWITCH:CONTIG -Allocate space contiguously [FALSE]
- SWITCH:PERMANENT -Allocate space permanently [FALSE]
- NUMBER:RLS=NRI/CR/CRE -Specifies the type of record level sharing (RLS) being done for a specific data set [NONE]
  - NRI -No Read Integrity
  - CR -Consistent Read
  - CRE -Consistent Read Explicit

4.2.163 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion to this CCSID [DEFAULT]
PATH:   CONV.WRITE.TEXT
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC
```

DESCRIPTION The TO code defines the encoding string or CCSID for the output text stream. The default value of this CCSID for read operations is UTF-8. For write operations, the default value is derived from the environment variable LANG. If LANG is not set, the default ASCII or EBCDIC system-codepage is used, depending on the platform.

To get a list of supported encoding strings, please use the command below:

```
   flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit. Optionally, you can use one of the keywords below to use the system-specific default ASCII or EBCDIC CCSID.

SELECTIONS

- `DEFAULT -Use default CCSID (system-specific)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`

4.2.164 PARAMETER *CASE*

SYNOPSIS

```
HELP:   Define upper, lower or special case conversion [NONE]
PATH:   CONV.WRITE.TEXT
TYPE:   NUMBER
SYNTAX: CASE=UPPER/LOWER/FOLD/SUPPER/SLOWER/USRTAB
```

DESCRIPTION With the case modifier, you can perform conversion to upper or lower case as well as apply case folding and some special case conversions.

Case folding converts all characters to lower case but is more compatible with language specific rules for doing so. For example, this may be useful for case-insensitive string comparison if the strings contain non-Latin characters (code point < 128).

This mapping is done for each code point in front of substitution or transliteration. The respective case is accomplished on the basis of a separate Unicode conversion table in memory. This conversion can be manipulated if user mapping is activated and mapping definitions ($\star$) are defined through the provided user table.

The options UPPER and LOWER convert characters without expansion, i.e. returned strings will never grow in length. Note, however, that case folding and the special options (convert e.g. ß to SS) can result in an expansion of data.

All these conversions are pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `UPPER -Change character to upper case`
- `LOWER -Change character to lower case`
- `FOLD -Change character to folded case (for compare)`
- `SUPPER -Change character to special upper case`
- `SLOWER -Change character to special lower case`
- `USRTAB -Change character according to user table`

4.2.165 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Write byte order mark in front of the output text [FALSE]
PATH:   CONV.WRITE.TEXT
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the write byte order mark keyword is defined, then the corresponding BOM sign is written at the start of the output stream (block oriented) or at the beginning of each record. If this BOM keyword is not used, then the BOM sign will not be written (this is valid independent of the chosen encoding/CCSID). If a valid BOM sign is located in the middle of the input stream, it is removed from the data. If you define the BOM keyword and the target charset contains only single bytes or no valid BOM sign, then no byte order mark will be written to the output file.

If an invalid BOM sign is recognized, then it will be ignored and causes no entry in the report file. If the byte order changes in the middle of text and the BOM handling switch is not activated, the conversion will stop and an error message is thrown.

By default, the write BOM keyword is set to OFF.

4.2.166 PARAMETER *ELF2NL*

SYNOPSIS

```
HELP:   Replace EBCDIC line feed (0x25) by new line (0x15) on output
PATH:   CONV.WRITE.TEXT
TYPE:   SWITCH
SYNTAX: ELF2NL
```

DESCRIPTION The replacement of EBCDIC line feed control characters (0x25) by new line characters (0x15) at conversion to EBCDIC can be activated on non EBCDIC (ASCII) systems or deactivated on EBCDIC systems by enabling this switch. This is for example required to prepare text files on open platforms for USS on z/OS. The usual delimiter for text in ASCII and UTF is 0x0A and will normally be translated to 0x25 in EBCDIC. On z/OS(USS), however, 0x15 (NL) is expected.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

To ensure a correctly delimited text record, only a line feed and not a carriage return (0x0D) line feed (0x0A) is used. If text formatting is used with method=NL, this will result in a single LF on distributed systems and the internal conversion will be activated by default to produce a valid text file for z/OS under USS on Windows, Unix and other open platforms.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.2.167 PARAMETER *CHRMODE*

SYNOPSIS

```
HELP:   Define handling of non-convertible characters [STOP]
PATH:   CONV.WRITE.TEXT
TYPE:   NUMBER
SYNTAX: CHRMODE=STOP/IGNORE/SUBSTITUTE
```

DESCRIPTION With the MODE selection you can define how an invalid or incomplete character is handled inside of a unit by this component. All possible modes are listed at the end of this chapter.

This MODE is valid for the content of each unit (block or record). At the end of a unit there can be an incomplete character. Such a rest will be handled dependent on the METHOD you choose. The rest of the last unit of a data field will be concatenated to the first unit of the next data field. For all other records in a data field the handling of such a rest will be defined by the respective METHOD.

The SKIP case for METHOD is only relevant for incomplete byte sequences at the end of a unit. The ignore and substitution mode is only effective for all characters in the middle of the unit, besides an incomplete byte sequence occurs in the last unit. Please don't mix up the IGNORE case for MODE with the SKIP case of METHOD.

If a system or user substitution table is used the stop, ignore or substitute are only relevant for characters not managed by the CCSIDs and the substitution tables. This means that the USRTAB and SYSTAB are added to the capabilities of character conversion and only for the rest the chosen MODE is used.

4.2.167.1 CONSTANT *STOP*

SYNOPSIS

```
HELP:   Stop at the first non-convertible character
PATH:   CONV.WRITE.TEXT.CHRMODE
TYPE:   NUMBER
SYNTAX: STOP
```

DESCRIPTION Stops conversion at the first incomplete or invalid character which can not be converted. An error message is given with the position of abortion and the value in the input stream. An entry will be (STP()) written to the report file to log this event. To be compatible with other conversion tools, this is the default case, but we recommend to use IGNORE or SUBSTITUTE to prevent any abortion.

indexterm:[Constant STOP}

4.2.167.2 CONSTANT *IGNORE*

SYNOPSIS

```
HELP:   Ignore non-convertible characters
PATH:   CONV.WRITE.TEXT.CHRMODE
TYPE:   NUMBER
SYNTAX: IGNORE
```

DESCRIPTION Ignores incomplete or invalid characters and writes a report entry for each character. IGNORE is a substitution by no corresponding character that means no substitution table is required.

indexterm:[Constant IGNORE}

4.2.167.3 CONSTANT *SUBSTITUTE*

SYNOPSIS

```
HELP:   Substitute non-convertible characters
PATH:   CONV.WRITE.TEXT.CHRMODE
TYPE:   NUMBER
SYNTAX: SUBSTITUTE
```

DESCRIPTION Substitutes incomplete or invalid character and writes a report entry for each character. For SUBSTITUTE you can define a substitution character list (SUBCHR). If no SUBCHR is provided then this option substitutes invalid or incomplete characters with code point 0x00001A. If a SUBCHR is defined all invalid or incomplete characters will be substituted by the defined list of code points that can be up to 8. If you specify a SUBCHR the substitution mode will be enabled.

indexterm:[Constant SUBSTITUTE}

4.2.168 PARAMETER *SUBCHAR*

SYNOPSIS

```
HELP:   Default substitution character list [0x1A] (maximal 8 code points)
PATH:   CONV.WRITE.TEXT
TYPE:   NUMBER
SYNTAX: SUBCHAR[num/SYSTEM...]
```

DESCRIPTION As substitution character (SUBCHR) you can define a list with up to 8 UNICODE points which are used for the substitution of invalid or incomplete characters if the corresponding code point is not defined in the substitution table (see SUBTAB). If you assign to it the keyword SYSTEM the value 0x1A is written as default substitution character.

If you activate substitution and no SUBCHR is defined, then unsupported characters will be replaced by CODEPOINT 0x00001A.

On the other hand SUBCHR will replace every undefined code point in the substitution table, what is the same behavior when using a USRTAB with corresponding SUBCHR values for every possible code point (000000-200000). If SUBCHR but no MODE is defined the substitution mode will be activated.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `SYSTEM -Use system specific substitution char (0x1A for ASCII/UTF, 0x3F for EBCDIC)`

4.2.169 PARAMETER *SYSTABLE*

SYNOPSIS

```
HELP:   Define a preloaded system substitution table
PATH:   CONV.WRITE.TEXT
TYPE:   NUMBER
SYNTAX: SYSTABLE=ICONV
```

DESCRIPTION With the system table (SYSTAB) you can load a substitution table with a predefined transliteration schema. Entries in the USRTAB can be used to override predefined transliteration entries or add missing transliteration code points. The options listed at the end of this chapter are available.

If a substitution table is used then the MODE option (STOP/IGNORE/SUBSTITUTE) is only relevant for characters that cannot be converted or substituted.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `ICONV -Preload with ICONV//TRANSLIT substitution table`

4.2.170 PARAMETER *USRTABLE*

SYNOPSIS

```
HELP:   Name of the file containing the user conversion table
PATH:   CONV.WRITE.TEXT
TYPE:   STRING
SYNTAX: USRTABLE='str'/NPAS/SEPA/DELA/DLAX
```

DESCRIPTION With this parameter, you can choose a predefined subset or standard manipulation for the system tables via keyword (see `flcl help "xcnv.inp .sav.fil.cnv.chr.usrtab"` for available keywords). Alternatively, you can use a file containing a user defined substitution table (USRTAB).

For the user table file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
USRTAB=NPAS                          ; use XOEV predefined subset
USRTAB=DELA                          ; German Latin single byte subset
USRTAB='~.MYUSRTAB(USRTAB1)'         ; PDS on z/OS
USRTAB='<SYSUID>.MYUSRT02'           ; PS dataset on z/OS
USRTAB='~/myusrtab3.txt'             ; Unix path name
USRTAB='<HOME>\usrtab\myusrtab4.txt' ; Windows path name
```

A user defined substitution table has the syntax listed below:

```
usr_tab_file         -> usr_definition_list
usr_definition_list  -> usr_definition usr_definition_list
                     |  EMPTY
usr_definition       -> '(' codepoint_definition ')'
codepoint_definition -> cp_activation
                     |  cp_deactivation
                     |  cp_transliteration
                     |  cp_standard_mapping
                     |  cp_usr_case_mapping
cp_activation        -> ['+'] CODEPOINT  [ '-' CODEPOINT ]
cp_deactivation      ->  '-'  CODEPOINT  [ '-' CODEPOINT ]
cp_transliteration   -> ['+'] CODEPOINT '=' code_point_list
cp_standard_mapping  ->  '*'  CODEPOINT '=' code_point_list
cp_usr_case_mapping  ->  '^'  CODEPOINT '=' code_point_list
code_point_list      -> CODEPOINT separator code_point_list
                     |  CODEPOINT
                     |  EMPTY
separator            -> '/'
                     |  ','
                     |  ';'
```

Note: In the description below "/" is used as code point separator.

The optional signs in front of a code point (CP) have the following meaning:

```
'+' - a valid transliteration (only if CP not in the target set)
'*' - a valid mapping         (this translation is always done)
'^' - a case mapping/folding  (only if case=usrtab is activated)
'-' - an invalid definition   (removes CP for subset definitions)
```

If no sign is used, then a transliteration for a valid code point is defined. Valid code points are accepted, invalid code points result in in an error. Error handling depends on the mode (STOP,IGNORE, SUBSTITUTE). Definition of + code points without code point list, thus without = is equal definition of valid code points. With an asterisk $*$, you can define an enforced mapping for this code point. For example, this can be used to delete this character if no code point list is provided or to translate this character always in another value. This can also be used to convert code points outside of a subset into this subset.

In contrast to the transliteration (+ is only done if a character doesn't exist in the target encoding), the mapping is always done for each character. Invalid (−) code points can be used to deactivate a code point. To activate (+) or deactivate (−) a range of code points you can use the minus mark (−) followed by one sub code point. The range goes from the smaller code point to the bigger one. For example:

```
(-00-7F) deactivate all US-ASCII code points
(+39-30) activate all decimal digits
```

The range operator (−) with the optional plus sign (+) can also be used to activate code points.

To define an own case mapping or folding (required for certain subsets) the caret sign (^) can be used. This mapping is only done if the user table is activated for case mapping (CASE=USRTAB).

To define transliterations/substitutions/mappings an assignment (=) of a code point list is required. The code point list can contain a maximum of 8 code points. If no code point list is specified, then the character is ignored. A transliteration to itself simply activates this code point. The same is valid for mapping definitions. A code point is a hexadecimal number representing a 21 bit UNICODE point.

```
   00000000 to 001FFFFF hex
```

If a code point is not defined in the substitution table (USRTAB and/or SYSTAB), the appropriate substitution character (SUBCHR) is used. If SUBCHR is not set, no substitution is performed. Depending on the MODE, character conversion stops with an error (STOP) or ignores the character (IGNORE).

Text before and after brackets are comments.

Between "(" and "=" or "/" or ")" you can define hex digits until the first non-hex digit. All non-hex digits up to the next separator are interpreted as comment. Leading whitespace is ignored.

```
   REPLACE GERMAN SZ (00DF=0073/0073)                  WITH ss
THIS IS AN EXAMPLE FOR COMMENTS IN CODE POINT LIST:
   REPLACE EURO MARK (20AC= 45#E / 55#U / 52#R / 4F#O) WITH EURO
   REPLACE BOM MARK  (EFFF=)                           WITH NOTHING
```

Leading zeros are allowed. If you don't supply a hex value, then 0x00 is used.

```
   REPLACE GERMAN SZ (00DF=/)                   WITH 0x00 0x00
   REPLACE EURO MARK (20AC=00000045/00055/000052/000004F) WITH EURO
```

ATTENTION: Please don't use parentheses "()" or the operators in your comments.

To describe your own subsets, you can use a user table without a system table. The USRTAB can also be used to overwrite or add transliterations when a system substitution table (for example SYSTAB=ICONV) is used. The transliteration works recursively, that is if one of the substitution code points is not in the target set, this substitution will be used instead and so on.

```
   REPLACE GERMAN OE (0000D6=004F/0045)       WITH O E
   REPLACE GERMAN SZ (0000DF=0073/0073)       WITH s s
   REPLACE EURO MARK (0020AC=00D6/00DF/52/4F) WITH OE SZ R O
```

If you have a EURO sign in your text and convert it to *Latin1*, the resulting byte string will be *D6DF524F*. On the other hand, if you convert it to *ASCII*, the byte string will be *4F457373524F*.

The mapping as described above is always done and is also done recursive including the transliteration result. For example, if you define a mapping `(*30=39)` then the target data won't have any zero in the resulting text.

By replacing other code points recursively, you could easily cause infinite loops. To prevent this, the amount of replacements and the length of one replacement is limited to a maximum of 64. Note that this could still result in a large expansion of data. Therefore, be careful about defining recursive substitutions.

A sample user table for the ICONV system table to change the transliteration of German umlauts to AE, OE or UE is located in the SAMPLE directory under CCUTDEXL. Another sample user table called CCUTNPAS defines the *string.latin* subset (XOEF) which is mainly used for statutory reporting.

The order of definition is the order of processing the definitions. For exampel, if you define a mapping or a transliteration for code point X and later you make this code point invalid, then the code point is invalid. On the other hand, you can first deactivate all code points, then activate your subset and define your transliterations (best fit) or mappings.

For example: To define a non-expansion best fit mapping a user table can be used to delete all combined characters for single byte code pages. For the ICONV system transliteration table, a sample user table can be found under the name CCUTBFM1.

Some other subset user table definitions were added. For example:

- currently valid SEPA characters (<128) for money transfers (CCUTSEPA)
- characters valid in ISO-8859-15, CP1252 and IBM1142 (CCUTDELA)
- characters valid in ISO-8859-15, CP1252, IBM1142 and XOEF (CCUTDLAX)

All user table definitions are pre-calculated once at the beginning of execution to reduce the required CPU time for the conversion. So, the use of a user table increases the cost to open a file but has no effect on CPU utilization when converting the data.

To minimize the effort and to simplify the usage of user table, the most common subsets can be selected by keyword. In this case, a pre-calculated load module (DLL) is used. The subset definition can be for UTF (21 Bit UNICODE) or UCS (16 Bit UNICODE). If it is a UCS subset, then no valid code point is greater than 0xFFFF and you must use UCS CCSIDS to ensure that no code point greater than 0xFFFF is accepted.

If you have a working user table definition for a popular subset or system table manipulation, you can request a keyword for it. Please report this requirement over our issue tracking system

```
http://www.flam.de/en/technology/support/issues/
```

and attach the corresponding user table definition.

SELECTIONS

- `NPAS -UCS subset for statutory reporting (New passport, XOEF, String.Latin)`
- `SEPA -UCS subset of valid SEPA character smaller then 128`
- `DELA -UCS subset for German Latin of IBM1141, ISO8859-15 and CP1252`
- `DLAX -UCS subset for German Latin of IBM1141, ISO8859-15, CP1252 and XOEF`

4.2.171 PARAMETER *REPORTFILE*

SYNOPSIS

```
HELP:   Name of the report file to log substitutions
PATH:   CONV.WRITE.TEXT
TYPE:   STRING
SYNTAX: REPORTFILE='str'/STDOUT/STDERR
```

DESCRIPTION With the report capability, you can log every substitution (stop / ignore / substitute / translit / skip / ...) to a log file. For the report file, all rules and replacements are valid, which are described under "FILENAME HANDLING" above. For example:

```
REPORT=STDERR                          ; is SYSPRINT on z/OS
REPORT='~.MYREPORT(REPORT1)'           ; PDS on z/OS
REPORT='<SYSUID>.MYREPRT2'             ; PS dataset on z/OS
REPORT='~/myreport3.txt'               ; Unix path name
REPORT='<HOME>\reports\myreport4.txt'  ; Windows path name
```

A substitution report has the syntax listed below:

```
report_file         -> sub_definition_list
sub_definition_list -> sub_definition sub_definition_list
                    |  EMPTY
sub_definition      -> SUB '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    -> XLT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    |  IGN '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  STP '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  IDT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  SKP '(' UNIT ':' OFFSET '+' POSITION ':'
```

```
                          code_point byte_string ')'
                    |  UKW '(' UNIT ':' OFFSET '+' POSITION ':'
                          code_point byte_string ')'
  code_point_list   -> CODEPOINT '/' code_point_list
                    |  EMPTY
  code_point        -> CODEPOINT
                    |  EMPTY
  byte_string       -> '(' BYTESTRING ')'
                    |  EMPTY
```

A comment behind brackets explains the reason for substitution:

```
'invalid character'
'incomplete character'
'incomplete character at the end of a record'
```

The notations STP (stop), IGN (ignore), SUB (substitute), XLT (transliteration), IDT (identical) or SKP (skip) before round brackets indicate the action taken. If the task can not be determined then UKW (unknown) is used. A ignore of an invalid character is realized as substitution with nothing. All substitutions with a length of 0 are indicated with IGN (ignore). The position of such an abnormal/irregular character is coded inside the braces:

- UNIT number of records or blocks (depends on the data)
- OFFSET byte offset up to this unit in the complete data stream
- POSITION number of bytes inside the unit where the activity was done

All counting starts with 0. The first block or record has unit 0.

The position is relative to the data stream which was presented to the character conversion module. After conversions and formatting the position values can be far off from the input file which was read originally. For example, when reading a text file with text formatting to records, then delimiting characters are not included in the byte offset, the units are records and the position indicates which character was the issue for this log entry.

The CODEPOINTs (UNICODE) are taken from the defined substitution table The BYTESTRINGs are taken from the input (read characters) and output data (written characters). BYTESTRINGs and CODEPOINTs are provided if they are known in the cases of such an irregularity. In some cases, these values can be missing.

For interpreting the report file, you must know that only the values inside round brackets are meaningful. Text before or behind round brackets are comments, but this may be subject to changes.

SELECTIONS

- `STDOUT -Write output to stdout`
- `STDERR -Write output to stderr`

4.2.172 OVERLAY *COMPRESS*

SYNOPSIS

```
HELP:   Compression procedure [NONE]
PATH:   CONV.WRITE.TEXT
TYPE:   OVERLAY
SYNTAX: COMPRESS/CMP.{GZIP()/BZIP2()/XZ()/FLAM4()}
```

DESCRIPTION With the overlay compress you can choose which kind of compression method is used for the data. Depending on the compression method different parameter can be provided to define the behavior in more detail.

If no compression method selected no compression of the data will be done.

See below the different possibilities of compression.

4.2.173 OBJECT *GZIP*

SYNOPSIS

```
HELP:   Compress data compliant to RCF1952 (GZIP)
PATH:   CONV.WRITE.TEXT.COMPRESS
TYPE:   OBJECT
SYNTAX: GZIP(LEVEL=num/COPY/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO, ↩
    COMMENT='str',CHECK)
```

DESCRIPTION The GZIP compression conforms to RCF1951/52. The compression level can be set to values from 0 to 9 (0=copy, 1=fastest, 9=best). Additionally, it is possible to set a comment and to enforce checksum calculation of the file header.

The checksum calculation is not recommended because only the newer versions of Zlib can handle this checksum technique. With older versions of Zlib this can cause problems. Therefore, checksum calculation is turned off by default.

The decompressor also accepts the older Zlib wrapping format specified in RFC1950. It is not possible to enforce RFC1950 at compression because most standards require RFC1952 for the deflate algorithms.

If no compression level is defined and the data is marked is being without redundancy, level 0 (copy) is used. In all other cases the default level 6 is used.

Decompression of concatenated GZIP files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

`filename/:memberindex`

ARGUMENTS

- `NUMBER:LEVEL=num/COPY/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level [AUTO==6]`
  - `COPY -No compression =copy of data (0)`
  - `FAST -Fastest compression (level 1)`
  - `LEV1 -Compression level 1`
  - `LEV2 -Compression level 2`
  - `LEV3 -Compression level 3`
  - `LEV4 -Compression level 4`
  - `LEV5 -Compression level 5`
  - `LEV6 -Compression level 6`
  - `LEV7 -Compression level 7`
  - `LEV8 -Compression level 8`
  - `LEV9 -Compression level 9`
  - `BEST -Best compression (level 9)`
  - `AUTO -Automatic compression (level 6)`
- `STRING:COMMENT='str' -Comment for header [NONE]`
- `SWITCH:CHECK -Calculate checksum over GZIP header [FALSE]`

4.2.174 OBJECT *BZIP2*

SYNOPSIS

```
HELP:   BZIP2 data compression technique
PATH:   CONV.WRITE.TEXT.COMPRESS
TYPE:   OBJECT
SYNTAX: BZIP2(LEVEL=num/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO)
```

DESCRIPTION The BZIP2 algorithm compresses chunks of data. Such a chunk can be up to 100KB, 200KB, . . . , 900KB. The chunk size is selected automatically based on the internal block size (block size in KB rounded up to the next possible chunk size).

The chunk size directly correlates with the compression level. Compression levels range from 1 (fastest) to 9 (best, but slowest). Level 1 results in 100KB chunk size, level 9 in 900KB chunk size (i.e. chunk size = level*100KB). Hence, a lower level also reduces the required memory resources during compression and decompression.

Decompression of concatenated BZIP2 files is also possible. All concatenated files will be decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
```

ARGUMENTS

- `NUMBER:LEVEL=num/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level (defines block size) [AUTO]`
  - `FAST -Fastest compression (Block size =100kb)`
  - `LEV1 -Compression level 1 (Block size =100kb)`
  - `LEV2 -Compression level 2 (Block size =200kb)`
  - `LEV3 -Compression level 3 (Block size =300kb)`
  - `LEV4 -Compression level 4 (Block size =400kb)`
  - `LEV5 -Compression level 5 (Block size =500kb)`
  - `LEV6 -Compression level 6 (Block size =600kb)`
  - `LEV7 -Compression level 7 (Block size =700kb)`
  - `LEV8 -Compression level 8 (Block size =800kb)`
  - `LEV9 -Compression level 9 (Block size =900kb)`
  - `BEST -Best compression (Block size =900kb)`
  - `AUTO -Automatic compression (depending on the real block size)`

4.2.175 OBJECT *XZ*

SYNOPSIS

```
HELP:   XZ (LZMA) data compression technique
PATH:   CONV.WRITE.TEXT.COMPRESS
TYPE:   OBJECT
SYNTAX: XZ(LEVEL=num/FAST/LEV0/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO)
```

DESCRIPTION The XZ compression conforms to the XZ-compression tool on UNIX-systems. The compression level can be set to values from 0 to 9 (0=fastest, 9=best).

The default level is 6 when no level is specified.

Decompression of concatenated XZ files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
```

ARGUMENTS

- `NUMBER:LEVEL=num/FAST/LEV0/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level [AUTO==6]`
  - `FAST -Fastest compression (level 0)`

- LEV0 -Compression level 0
- LEV1 -Compression level 1
- LEV2 -Compression level 2
- LEV3 -Compression level 3
- LEV4 -Compression level 4
- LEV5 -Compression level 5
- LEV6 -Compression level 6
- LEV7 -Compression level 7
- LEV8 -Compression level 8
- LEV9 -Compression level 9
- BEST -Best compression (level 9)
- AUTO -Automatic compression (level 6)

4.2.176 OBJECT *FLAM4*

SYNOPSIS

```
HELP:   Compress data with FLAM
PATH:   CONV.WRITE.TEXT.COMPRESS
TYPE:   OBJECT
SYNTAX: FLAM4(MODE=CX7/CX8/VR8/ADC/NDC,COMMENT='str')
```

DESCRIPTION The FLAM4 compression to write compressed data into FLAMFILEs. As main argument you can define the compression mode (ADC, VR8, CX8). Additionally, it is possible to set a comment in the file header.

If the record-oriented method (write.record()) is used, the records with all attributes are written to the FLAM archive. If a block-oriented method (write.binary/char/text/xml()) is used, the data blocks are wrapped into records and written to the FLAMFILE as binary chunks with an undefined record format (U).

ARGUMENTS

- NUMBER:MODE=CX7/CX8/VR8/ADC/NDC -Compression mode [ADC]
  - CX7 -Compression encoding as printable characters
  - CX8 -Compression encoding as binary data
  - VR8 -Enhanced compression encoding as binary data
  - ADC -Advanced Data Compression
  - NDC -No data compression
- STRING:COMMENT='str' -Comment for the member in the FLAMFILE [internal use]

4.2.177 OVERLAY *ENCRYPT*

SYNOPSIS

```
HELP:   Encryption procedure [NONE]
PATH:   CONV.WRITE.TEXT
TYPE:   OVERLAY
SYNTAX: ENCRYPT/ENC.{F4PWD()/F4KME()}
```

DESCRIPTION The overlay *encrypt* allows you to choose which kind of encryption method is used for the data. Depending on the encryption method, different parameters can be set.

If no encryption method is selected, then no data encryption will be done.

See below for the encryption possibilities.

4.2.178 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password encryption
PATH:   CONV.WRITE.TEXT.ENCRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(MODE=AES/FLAM,PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Activates FLAM4 password-based encryption to encrypt the data written to FLAMFILEs. You can choose the encryption mode. The default and recommended mode is AES. If you provide no password, a hardcoded default password is used.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

If the data is already compressed, then MODE=NDC (no data compression) is used. Otherwise, ADC compression is used. If you set the compression mode to CX7/CX8/VR8 and enable encryption, e.g.

```
  comp.flam(mode=CX7/CX8/VR8) encr.f4pwd()
```

the compression mode is ignored and ADC/NDC is used instead. Due to how those three old compression modes work, they are incompatible with FLAM encryption.

ARGUMENTS

- NUMBER:MODE=AES/FLAM -Encryption mode [AES]
  - AES -AES encryption
  - FLAM -FLAM encryption
- STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']
  - DEFAULT -FLAM4 default password

4.2.179 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   CONV.WRITE.TEXT.ENCRYPT
TYPE:   OBJECT
SYNTAX: F4KME(MODE=AES/FLAM,LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 encryption of data written to FLAMFILEs based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME parameter is provided (encr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using encr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

You may define the encryption mode. The default and recommended mode is AES.

If the data is already compressed, then MODE=NDC (no data compression) is used. Otherwise, ADC compression is used to create the FLAMFILE. If you set the compression mode to CX7/CX8/VR8 and enable encryption, e.g.

```
comp.flam(mode=CX7/CX8/VR8) encr.f4kme()
```

the compression mode is ignored and ADC/NDC is used instead. Due to how those three old compression modes work, they are incompatible with FLAM encryption.

ARGUMENTS

- `NUMBER:MODE=AES/FLAM -Encryption mode [AES]`
  - `AES -AES encryption`
  - `FLAM -FLAM encryption`
- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME [”]`

4.2.180 OVERLAY *ENCODE*

SYNOPSIS

```
HELP:   Encoding procedure [NONE]
PATH:   CONV.WRITE.TEXT
TYPE:   OVERLAY
SYNTAX: ENCODE/COD.{BASE64()/BASE32()/BASE16()/OPGP()}
```

DESCRIPTION The overlay *encode* allows to choose which kind of encoding method is used for the data. Depending on the encoding method, different parameters can be set.

If no encoding method is selected, then no data encoding will be done.

See below for the available encoding options.

4.2.181 OBJECT *BASE64*

SYNOPSIS

```
HELP:   Base64 encoding (RFC4648)
PATH:   CONV.WRITE.TEXT.ENCODE
TYPE:   OBJECT
SYNTAX: BASE64(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)

  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.182 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.TEXT.ENCODE.BASE64
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.183 OBJECT *BASE32*

SYNOPSIS

```
HELP:   Base32 encoding (RFC4648)
PATH:   CONV.WRITE.TEXT.ENCODE
TYPE:   OBJECT
SYNTAX: BASE32(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.184 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.TEXT.ENCODE.BASE32
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
```

```
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.185 OBJECT *BASE16*

SYNOPSIS

```
HELP:   Base16 encoding (RFC4648)
PATH:   CONV.WRITE.TEXT.ENCODE
TYPE:   OBJECT
SYNTAX: BASE16(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)

  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)

- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.186 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.TEXT.ENCODE.BASE16
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.187 OBJECT *OPGP*

SYNOPSIS

```
HELP:   Base64 encoding with 76 character, CRC checksum and Armor clasp (RFC4880)
PATH:   CONV.WRITE.TEXT.ENCODE
TYPE:   OBJECT
SYNTAX: OPGP(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE ↩
    /UTF32BE/UCS4LE/UTF32LE,DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM)
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- `NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]`
  - `NONE -No character set defined`
  - `SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)`
  - `ASCII -ASCII (mainly used in the for open system)`
  - `UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)`
  - `UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)`
  - `EBCDIC -EBCDIC (mainly used on IBM mainframe)`
  - `UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)`
  - `UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)`
  - `UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)`
  - `UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)`
  - `UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)`

  - UTF32BE –UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE –UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE –UTF-32 Little Endian (multibyte character set < 2M)

- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM –Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST –Adds no delimiter (HOST)
  - BIN –Adds no delimiter (HOST)
  - NL –Adds delimiter new line (USS)
  - USS –Adds delimiter for USS (new line)
  - LF –Adds delimiter line feed (UNIX)
  - UNIX –Adds delimiter for UNIX (line feed)
  - CR –Adds delimiter carriage return (MAC)
  - OLDMAC –Adds delimiter for old MACs (carriage return)
  - CRLF –Adds delimiter carriage return line feed (WIN)
  - WINDOWS –Adds delimiter for WINDOWS (carriage return line feed)
  - DLM –Adds the system specific delimiter (DEFAULT)
  - SYSTEM –Adds the system specific delimiter (DEFAULT)

4.2.188 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   CONV.WRITE.TEXT
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control character handling is a special feature for record oriented files of mainframe system. For print control, the ASA or machine control charaters are encoded in the first byte of the record.

If an attribute with such a control character is known on write operation, you can decide whether it will be part of the output (RETAIN), written to the output if required (DETACH) or never written (ERASE).

SELECTIONS

- DETACH –Manage print control character as record attribute
- RETAIN –Leave print control character as part of the data record
- ERASE –Erase the print control character from the record

4.2.189 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for write operation
PATH:   CONV.WRITE.TEXT
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.2.190 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for write operation
PATH:   CONV.WRITE.TEXT
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.2.191 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of output data
PATH:   CONV.WRITE.TEXT
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
OUTPUT=STDERR                          ; is SYSPRINT on z/OS
OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
OUTPUT='~/mycheck3.txt'                ; Unix path name
OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.2.192 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   CONV.WRITE.TEXT.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- `STRING:VALUE='str' -Checksum as hex-string`
- `STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`

4.2.193 OBJECT *XML*

SYNOPSIS

```
HELP:   Write XML data to a file
PATH:   CONV.WRITE
TYPE:   OBJECT
SYNTAX: XML(FILE='str'/STREAM/DUMMY,FALLOC(),REMAIN,APPEND,FLUSH,METHOD.{},CCSID='str'/ ↩
    DEFAULT/ASCII/EBCDIC,BOM,ELF2NL,CHRMODE=STOP/IGNORE/SUBSTITUTE,SUBCHAR[num/SYSTEM...], ↩
    SYSTABLE=ICONV,USRTABLE='str'/NPAS/SEPA/DELA/DLAX,REPORTFILE='str'/STDOUT/STDERR, ↩
    COMPRESS.{},ENCRYPT/ENC.{},ENCODE/COD.{},FLAM4NDC,FRCREC,LANG='str',PLATFORM=WIN/UNX/ZOS ↩
    /USS/VSE/BS2/MAC,HASH())
```

DESCRIPTION Write XML formats FLAM elements containing parsed XML data. The output is a valid XML document. The output XML can be formatted using 3 different methods:

- STANDARD: The XML document is reconstructed as close to the original XML document as possible.
- PRETTYPRINT: Creates an XML document with indentation suitable for human readability.
- MINIMIZED: Creates a minimized XML document. Line breaks and whitespace are removed unless they are part of actual character data.

The XML output is written in blocks and encoded in UTF-8 with line feed (0xA) as line delimiter unless a CCSID is supplied, in which case character conversion is performed.

If writing XML, the semantics of read modes (read.*()) change as follows:

- BINARY: The binary input data is assumed to be a valid XML document encoded in UTF-8. It is parsed and then written according to the selected write options.
- CHARACTER: The input data is assumed to be a valid XML document. The input is converted to UTF-8 before it is passed to the XML parser. The parsed XML data is then written according to the selected write options.
- TEXT: The input data is assumed to be a valid XML document. The input is converted to UTF-8 before it is passed to the XML parser. The parsed XML data is then written according to the selected write options.
- XML: The input data is a valid XML document. If a CCSID is supplied, the input is converted to UTF-8 prior to parsing. If no CCSID is given, then UTF-8 is assumed.
- RECORD: The input records are converted to UTF-8, line delimiters added after each record and converted into binary blocks before the input is passed to the XML parser. The parsed XML data is then written according to the selected write options.
- FLAM4: Same as RECORD, but read from a FLAM4 file.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to write ["==origin.ext]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`
- `SWITCH:REMAIN -Remain existing files (don't overwrite) [FALSE]`
- `SWITCH:APPEND -Append data to this file [FALSE]`
- `SWITCH:FLUSH -Enforce flush in front of close file [FALSE]`
- `SWITCH:FLAM4NDC -Write data as FLAM4FILE in MODE=NDC(No Data Compression) [FALSE]`
- `SWITCH:FRCREC -Enforce record orientation on record oriented devices [FALSE]`

4.2.194 OBJECT *FALLOC*

SYNOPSIS

```
HELP:   Platform dependent file allocation parameter
PATH:   CONV.WRITE.XML
TYPE:   OBJECT
SYNTAX: FALLOC(ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZE=num,RECFORMAT=NONE/BIN/TXT/ ↩
    DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX- ↩
    REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR ↩
    /FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/ ↩
    VRA/VRM,RECLENGTH=num,KEYDISP=OLD/NEW/DEL,KEYPOSITION=num,KEYLENGTH=num,SUBSYSTEM(), ↩
    SPACE(),LIKE='str',RENEW)
```

DESCRIPTION The file allocation object collects several parameters for platforms which are record oriented based on a catalog system like z/OS® and where the space, volume and other things must be predefined. The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

But also on distributed platforms like (WIN, UNIX, MAC, ...) FLAM supports record oriented file types to keep the record information on such platforms. For this case the different record format for the open world can be defined. To read such a special file format on an open platform the correct record format must also be defined. On a catalog based system, this is not required. If the record format in the catalog are not the same as specified an error will occur.

ARGUMENTS

- NUMBER:ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [SEQ]
  - SEQ -Sequential data set [DEFAULT]
  - LIB -Library (ZOS-PDSE)
  - PDS -Partitioned data set (ZOS-PDS)
  - USS -UNIX files (path based)
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)
- NUMBER:BLKSIZE=num -Block size [system]
- NUMBER:RECFORMAT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format [auto]
  - NONE -No format defined
  - BIN -Open -Binary (undefined) data (only the last record are shorter then record length)
  - TXT -Open -Text data with standard delimiter (result is a stripped variable length records)
  - DLM -Open -Records separated by a special delimiter with the result of variable length records
  - VAR -Open -Variable length record (if necessary define length format)
  - VAR-ASA -Open -Variable length record with ASA print control character (if necessary define length format)
  - VAR-MCC -Open -Variable length record with machine print control codes (if necessary define length format)
  - FIX -Open -Fix length record (file size =N * record length)
  - FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
  - FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
  - VAR-REL -Open -Variable relative record (if necessary define length format)
  - VAR-REL-ASA -Open -Variable relative record with ASA print control character (if necessary define length format)
  - VAR-REL-MCC -Open -Variable relative record with machine print control codes (if necessary define length format)
  - FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
  - FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size = N * (record length + 9))
  - FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size = N * (record length + 9))
  - F -Host -Fixed-length, unblocked
  - FA -Host -Fixed-length, unblocked, ASA print control characters
  - FM -Host -Fixed-length, unblocked, machine print control codes
  - FS -Host -Fixed-length, standard
  - FSA -Host -Fixed-length, standard, ASA print control characters
  - FSM -Host -Fixed-length, standard, machine print control codes

  - FB -Host -Fixed-length, blocked
  - FBA -Host -Fixed-length, blocked, ASA print control characters
  - FBM -Host -Fixed-length, blocked, machine print control codes
  - FBS -Host -Fixed-length, blocked, standard
  - FBSA -Host -Fixed-length, blocked, standard, ASA print control characters
  - FBSM -Host -Fixed-length, blocked, standard, machine print control codes
  - FE -Host -Fixed-length, ESDS
  - FEA -Host -Fixed-length, ESDS, ASA print control characters
  - FEM -Host -Fixed-length, ESDS, machine print control codes
  - FK -Host -Fixed-length, KSDS
  - FKA -Host -Fixed-length, KSDS, ASA print control characters
  - FKM -Host -Fixed-length, KSDS, machine print control codes
  - FR -Host -Fixed-length, RRDS
  - FRA -Host -Fixed-length, RRDS, ASA print control characters
  - FRM -Host -Fixed-length, RRDS, machine print control codes
  - U -Host -Undefined-length
  - UA -Host -Undefined-length, ASA print control characters
  - UM -Host -Undefined-length, machine print control codes
  - V -Host -Variable
  - VA -Host -Variable, ASA print control characters
  - VM -Host -Variable, machine print control codes
  - VS -Host -Variable, spanned
  - VSA -Host -Variable, spanned, ASA print control characters
  - VSM -Host -Variable, spanned, machine print control codes
  - VB -Host -Variable, blocked
  - VBA -Host -Variable, blocked, ASA print control characters
  - VBM -Host -Variable, blocked, machine print control codes
  - VBS -Host -Variable, blocked, spanned
  - VBSA -Host -Variable, blocked, spanned, ASA print control characters
  - VBSM -Host -Variable, blocked, spanned, machine print control codes
  - VE -Host -Variable, ESDS
  - VEA -Host -Variable, ESDS, ASA print control characters
  - VEM -Host -Variable, ESDS, machine print control codes
  - VK -Host -Variable, KSDS
  - VKA -Host -Variable, KSDS, ASA print control characters
  - VKM -Host -Variable, KSDS, machine print control codes
  - VR -Host -Variable, RRDS
  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes

- NUMBER:RECLENGTH=num -Record length (data only, no print control and length fields)
- NUMBER:KEYDISP=OLD/NEW/DEL -Key disposition for index/key sequential data sets [OLD]
  - OLD -Key disposition OLD (keep the existing key)
  - NEW -Key disposition NEW (insert record number as key)

  - DEL -Key disposition DEL (delete key from record)

- NUMBER:KEYPOSITION=num -Key position (>=1) for index/key sequential data sets [1]
- NUMBER:KEYLENGTH=num -Key length (>=1) for index/key sequential data sets [8]
- STRING:LIKE='str' -Specifies the name of the model data set from which the attributes are to be copied
- SWITCH:RENEW -Enforce new allocation by a remove of an existing file

4.2.195 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   CONV.WRITE.XML.FALLOC
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- STRING:NAME='str' -Name of the subsystem
- STRING:PARM='str' -Parameter list for the subsystem (comma separated)

4.2.196 OBJECT *SPACE*

SYNOPSIS

```
HELP:   Platform dependent space definition parameter
PATH:   CONV.WRITE.XML.FALLOC
TYPE:   OBJECT
SYNTAX: SPACE(PRIMARY=num,SECONDARY=num,DIRBLOCKS=num,VOLSER='str',VOLUNIT='str',DATACLASS ↩
    ='str',MGMTCLASS='str',STORCLASS='str',NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG, ↩
    ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG,RELEASE=YES/NO,UNMOVABLE,CONTIG,PERMANENT,RLS= ↩
    NRI/CR/CRE)
```

DESCRIPTION The space definition object collects several parameters for platforms where data sets are preallocated like z/OS® and where clylinders or tracks must be predefined for a file allocation.

The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

All size values must be given in megabyte (MiB). The calculation in the platform dependent units (tracks, cylinders, ...) are done inernally. Note: 1 MiB = 1024 KiB = 1024*1024 Byte

ARGUMENTS

- NUMBER:PRIMARY=num -Primary size in megabytes
- NUMBER:SECONDARY=num -Secondary size in megabytes
- NUMBER:DIRBLOCKS=num -Directory blocks for partitioned data sets
- STRING:VOLSER='str' -Volume serial number
- STRING:VOLUNIT='str' -Volume unit
- STRING:DATACLASS='str' -Data class
- STRING:MGMTCLASS='str' -Management class
- STRING:STORCLASS='str' -Storage class
- NUMBER:NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for normal end [CATALOG]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for abnormal end [DELETE]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:RELEASE=YES/NO -Release unused space when file is closed [SEQ=YES, PDS=NO]
  - YES -Yes, do it
  - NO -No, don't do it
- SWITCH:UNMOVABLE -Allocate space unmovable [FALSE]
- SWITCH:CONTIG -Allocate space contiguously [FALSE]
- SWITCH:PERMANENT -Allocate space permanently [FALSE]
- NUMBER:RLS=NRI/CR/CRE -Specifies the type of record level sharing (RLS) being done for a specific data set [NONE]
  - NRI -No Read Integrity
  - CR -Consistent Read
  - CRE -Consistent Read Explicit

4.2.197 OVERLAY *METHOD*

SYNOPSIS

```
HELP:   Method for XML formatting [STANDARD]
PATH:   CONV.WRITE.XML
TYPE:   OVERLAY
SYNTAX: METHOD.{STANDARD()/PRETTYPRINT()/MINIMIZED()}
```

DESCRIPTION The XML formatting method defines how an XML document will be created and written from FLAM elements containing XML data. It is also possible to dump non-XML elements into an XML document in raw format.

The supported methods are listed below.

The standard method is the default for *write.binary()* and *write.xml()*. Minimized is the default for *write.char()*. Pretty printing is the default for *write.record()* and *write.flam()*.

4.2.198 OBJECT *STANDARD*

SYNOPSIS

```
HELP:   Reproduce all XML elements as is
PATH:   CONV.WRITE.XML.METHOD
TYPE:   OBJECT
SYNTAX: STANDARD(NOCMNT,BUFSIZ=num,INICNT=num)
```

DESCRIPTION The standard method recreates the original XML document from FLAM XML elements as close as possible. All document structure, indentation, line breaks, etc. are preserved. All line breaks are normalized to line feeds (0xA) according to the XML specification.

ARGUMENTS

- `SWITCH:NOCMNT -Do not write out XML comments [FALSE]`
- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]`
- `NUMBER:INICNT=num -Initial amount of records for preallocation [128]`

4.2.199 OBJECT *PRETTYPRINT*

SYNOPSIS

```
HELP:   Indentation of XML tags for human readability
PATH:   CONV.WRITE.XML.METHOD
TYPE:   OBJECT
SYNTAX: PRETTYPRINT(MAXWSP=num,WSPERR=WRITE/ABORT,INDSIZ=num,INDCHR=SPACE/TABULATOR,NOCMNT, ↩
    BUFSIZ=num,INICNT=num)
```

DESCRIPTION The pretty printing method creates an XML document that is very suitable to be read by humans. All XML elements start on its own line and are properly indented occording to its hierarchical position, unless they are surrounded by text data. Elements containing actual text (i.e. they do not only contain whitespace and newlines) have all their whitespace and newline characters preserved.

Example: If the original XML document is the following one:

```
<?xml version="1.0"?>
     <!DOCTYPE root [
  <!ELEMENT root (someelement)>
          ]>
  <root>
                <somelement   attribute1="value1" attribute2="value2">
```

```
    some text
   </somelement>
        <somelement>
    <!-- comment -->
           text in non-root element    <leaf>this is a leaf</leaf></somelement></root>
```

then pretty printing will result in the following document:

```
<?xml version="1.0"?>
<!DOCTYPE root [
  <!ELEMENT root (someelement)>
]>
<root>
  <somelement attribute1="value1" attribute2="value2">
    some text
   </somelement>
  <somelement>
    <!-- comment -->
           text in non-root element    <leaf>this is a leaf</leaf>
  </somelement>
</root>
```

ARGUMENTS

- NUMBER:MAXWSP=num -Size of internal whitespace cache (only used when method is minimize or pretty printing) [4096]
- NUMBER:WSPERR=WRITE/ABORT -Action when the whitespace cache overflows [WRITEOUT]
  - WRITE -Write out cached whitespace on full buffer, causing less nicely formatted output
  - ABORT -Abort XML writing when the whitespace buffer is full, output formatting has priority
- NUMBER:INDSIZ=num -Number of indentation characters per indentation level [2 (SPACE) or 1 (TAB)]
- NUMBER:INDCHR=SPACE/TABULATOR -Indentation character used to indent XML tags [SPACE]
  - SPACE -Use the space character as indentation character
  - TABULATOR -Use the tabulator character as indentation character
- SWITCH:NOCMNT -Do not write out XML comments [FALSE]
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of elements for preallocation [128]

4.2.200 OBJECT *MINIMIZED*

SYNOPSIS

```
HELP:   Discards comments, insignificant whitespace and whitespace-only sections between ↩
    tags
PATH:   CONV.WRITE.XML.METHOD
TYPE:   OBJECT
SYNTAX: MINIMIZED(MAXWSP=num,WSPERR=WRITE/ABORT,BUFSIZ=num,INICNT=num)
```

DESCRIPTION The minimized method creates an XML document that has whitespace and linebreaks removed unless they are part of other character data, as well as all comments. This is most suitable if data is primarily contained in the child nodes of the XML document, the document is read by a machine and if storage/bandwidth is an issue.

Example: If the original XML document is the following one:

```
<?xml version="1.0"?>
  <!DOCTYPE root [
    <!ELEMENT root (someelement)>
  ]>
<root>
  <somelement   attribute1="value1" attribute2="value2">
    some text
  </somelement>
  <somelement>
    <!-- comment -->
    text in non-root element
    <leaf>this is a leaf</leaf>
  </somelement>
</root>
```

then minimization will result in the following document:

```
<?xml version="1.0"?><!DOCTYPE root [<!ELEMENT root (someelement)>]><root><somelement  ↩
    attribute1="value1" attribute2="value2">
    some text
  </somelement><somelement>

    text in non-root element
    <leaf>this is a leaf</leaf></somelement></root>
```

ARGUMENTS

- NUMBER:MAXWSP=num -Size of internal whitespace cache (only used when method is minimize or pretty printing) [4096]
- NUMBER:WSPERR=WRITE/ABORT -Action when the whitespace cache overflows [WRITEOUT]
  - WRITE -Write out cached whitespace on full buffer, causing less nicely formatted output
  - ABORT -Abort XML writing when the whitespace buffer is full, output formatting has priority
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of elements for preallocation [128]

4.2.201 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion to this CCSID [no character conversion, leave in UTF-8]
PATH:   CONV.WRITE.XML
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC
```

DESCRIPTION The TO code defines the encoding string or CCSID for the output text stream. The default value of this CCSID for read operations is UTF-8. For write operations, the default value is derived from the environment variable LANG. If LANG is not set, the default ASCII or EBCDIC system-codepage is used, depending on the platform.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit. Optionally, you can use one of the keywords below to use the system-specific default ASCII or EBCDIC CCSID.

SELECTIONS

- `DEFAULT -Use default CCSID (system-specific)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`

4.2.202 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Write byte order mark in front of the output text [FALSE]
PATH:   CONV.WRITE.XML
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the write byte order mark keyword is defined, then the corresponding BOM sign is written at the start of the output stream (block oriented) or at the beginning of each record. If this BOM keyword is not used, then the BOM sign will not be written (this is valid independent of the chosen encoding/CCSID). If a valid BOM sign is located in the middle of the input stream, it is removed from the data. If you define the BOM keyword and the target charset contains only single bytes or no valid BOM sign, then no byte order mark will be written to the output file.

If an invalid BOM sign is recognized, then it will be ignored and causes no entry in the report file. If the byte order changes in the middle of text and the BOM handling switch is not activated, the conversion will stop and an error message is thrown.

By default, the write BOM keyword is set to OFF.

4.2.203 PARAMETER *ELF2NL*

SYNOPSIS

```
HELP:   Replace EBCDIC line feed (0x25) by new line (0x15) on output
PATH:   CONV.WRITE.XML
TYPE:   SWITCH
SYNTAX: ELF2NL
```

DESCRIPTION The replacement of EBCDIC line feed control characters (0x25) by new line characters (0x15) at conversion to EBCDIC can be activated on non EBCDIC (ASCII) systems or deactivated on EBCDIC systems by enabling this switch. This is for example required to prepare text files on open platforms for USS on z/OS. The usual delimiter for text in ASCII and UTF is 0x0A and will normally be translated to 0x25 in EBCDIC. On z/OS(USS), however, 0x15 (NL) is expected.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

To ensure a correctly delimited text record, only a line feed and not a carriage return (0x0D) line feed (0x0A) is used. If text formatting is used with method=NL, this will result in a single LF on distributed systems and the internal conversion will be activated by default to produce a valid text file for z/OS under USS on Windows, Unix and other open platforms.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.2.204 PARAMETER *CHRMODE*

SYNOPSIS

```
HELP:   Define handling of non-convertible characters [STOP]
PATH:   CONV.WRITE.XML
TYPE:   NUMBER
SYNTAX: CHRMODE=STOP/IGNORE/SUBSTITUTE
```

DESCRIPTION With the MODE selection you can define how an invalid or incomplete character is handled inside of a unit by this component. All possible modes are listed at the end of this chapter.

This MODE is valid for the content of each unit (block or record). At the end of a unit there can be an incomplete character. Such a rest will be handled dependent on the METHOD you choose. The rest of the last unit of a data field will be concatenated to the first unit of the next data field. For all other records in a data field the handling of such a rest will be defined by the respective METHOD.

The SKIP case for METHOD is only relevant for incomplete byte sequences at the end of a unit. The ignore and substitution mode is only effective for all characters in the middle of the unit, besides an incomplete byte sequence occurs in the last unit. Please don't mix up the IGNORE case for MODE with the SKIP case of METHOD.

If a system or user substitution table is used the stop, ignore or substitute are only relevant for characters not managed by the CCSIDs and the substitution tables. This means that the USRTAB and SYSTAB are added to the capabilities of character conversion and only for the rest the chosen MODE is used.

4.2.204.1 CONSTANT *STOP*

SYNOPSIS

```
HELP:   Stop at the first non-convertible character
PATH:   CONV.WRITE.XML.CHRMODE
TYPE:   NUMBER
SYNTAX: STOP
```

DESCRIPTION Stops conversion at the first incomplete or invalid character which can not be converted. An error message is given with the position of abortion and the value in the input stream. An entry will be (STP()) written to the report file to log this event. To be compatible with other conversion tools, this is the default case, but we recommend to use IGNORE or SUBSTITUTE to prevent any abortion.

indexterm:[Constant STOP}

4.2.204.2 CONSTANT *IGNORE*

SYNOPSIS

```
HELP:   Ignore non-convertible characters
PATH:   CONV.WRITE.XML.CHRMODE
TYPE:   NUMBER
SYNTAX: IGNORE
```

DESCRIPTION Ignores incomplete or invalid characters and writes a report entry for each character. IGNORE is a substitution by no corresponding character that means no substitution table is required.

indexterm:[Constant IGNORE}

4.2.204.3 CONSTANT *SUBSTITUTE*

SYNOPSIS

```
HELP:   Substitute non-convertible characters
PATH:   CONV.WRITE.XML.CHRMODE
TYPE:   NUMBER
SYNTAX: SUBSTITUTE
```

DESCRIPTION Substitutes incomplete or invalid character and writes a report entry for each character. For SUBSTITUTE you can define a substitution character list (SUBCHR). If no SUBCHR is provided then this option substitutes invalid or incomplete characters with code point 0x00001A. If a SUBCHR is defined all invalid or incomplete characters will be substituted by the defined list of code points that can be up to 8. If you specify a SUBCHR the substitution mode will be enabled.

indexterm:[Constant SUBSTITUTE}

4.2.205 PARAMETER *SUBCHAR*

SYNOPSIS

```
HELP:   Default substitution character list [0x1A] (maximal 8 code points)
PATH:   CONV.WRITE.XML
TYPE:   NUMBER
SYNTAX: SUBCHAR[num/SYSTEM...]
```

DESCRIPTION As substitution character (SUBCHR) you can define a list with up to 8 UNICODE points which are used for the substitution of invalid or incomplete characters if the corresponding code point is not defined in the substitution table (see SUBTAB). If you assign to it the keyword SYSTEM the value 0x1A is written as default substitution character.

If you activate substitution and no SUBCHR is defined, then unsupported characters will be replaced by CODEPOINT 0x00001A.

On the other hand SUBCHR will replace every undefined code point in the substitution table, what is the same behavior when using a USRTAB with corresponding SUBCHR values for every possible code point (000000-200000). If SUBCHR but no MODE is defined the substitution mode will be activated.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `SYSTEM -Use system specific substitution char (0x1A for ASCII/UTF, 0x3F for EBCDIC)`

4.2.206 PARAMETER *SYSTABLE*

SYNOPSIS

```
HELP:   Define a preloaded system substitution table
PATH:   CONV.WRITE.XML
TYPE:   NUMBER
SYNTAX: SYSTABLE=ICONV
```

DESCRIPTION With the system table (SYSTAB) you can load a substitution table with a predefined transliteration schema. Entries in the USRTAB can be used to override predefined transliteration entries or add missing transliteration code points. The options listed at the end of this chapter are available.

If a substitution table is used then the MODE option (STOP/IGNORE/SUBSTITUTE) is only relevant for characters that cannot be converted or substituted.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `ICONV -Preload with ICONV//TRANSLIT substitution table`

4.2.207 PARAMETER *USRTABLE*

SYNOPSIS

```
HELP:   Name of the file containing the user conversion table
PATH:   CONV.WRITE.XML
TYPE:   STRING
SYNTAX: USRTABLE='str'/NPAS/SEPA/DELA/DLAX
```

DESCRIPTION With this parameter, you can choose a predefined subset or standard manipulation for the system tables via keyword (see `flcl help "xcnv.inp .sav.fil.cnv.chr.usrtab"` for available keywords). Alternatively, you can use a file containing a user defined substitution table (USRTAB).

For the user table file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   USRTAB=NPAS                          ; use XOEV predefined subset
   USRTAB=DELA                          ; German Latin single byte subset
   USRTAB='~.MYUSRTAB(USRTAB1)'         ; PDS on z/OS
   USRTAB='<SYSUID>.MYUSRT02'           ; PS dataset on z/OS
   USRTAB='~/myusrtab3.txt'             ; Unix path name
   USRTAB='<HOME>\usrtab\myusrtab4.txt' ; Windows path name
```

A user defined substitution table has the syntax listed below:

```
   usr_tab_file         -> usr_definition_list
   usr_definition_list  -> usr_definition usr_definition_list
                        |  EMPTY
   usr_definition       -> '(' codepoint_definition ')'
   codepoint_definition -> cp_activation
                        |  cp_deactivation
                        |  cp_transliteration
                        |  cp_standard_mapping
                        |  cp_usr_case_mapping
   cp_activation        -> ['+'] CODEPOINT  [ '-' CODEPOINT ]
   cp_deactivation      ->  '-'  CODEPOINT  [ '-' CODEPOINT ]
   cp_transliteration   -> ['+'] CODEPOINT '=' code_point_list
   cp_standard_mapping  ->  '*'  CODEPOINT '=' code_point_list
   cp_usr_case_mapping  ->  '^'  CODEPOINT '=' code_point_list
   code_point_list      -> CODEPOINT separator code_point_list
                        |  CODEPOINT
                        |  EMPTY
   separator            -> '/'
                        |  ','
                        |  ';'
```

Note: In the description below "/" is used as code point separator.

The optional signs in front of a code point (CP) have the following meaning:

```
   '+' - a valid transliteration (only if CP not in the target set)
   '*' - a valid mapping         (this translation is always done)
   '^' - a case mapping/folding  (only if case=usrtab is activated)
   '-' - an invalid definition   (removes CP for subset definitions)
```

If no sign is used, then a transliteration for a valid code point is defined. Valid code points are accepted, invalid code points result in in an error. Error handling depends on the mode (STOP,IGNORE, SUBSTITUTE). Definition of + code points without code point list, thus without = is equal definition of valid code points. With an asterisk *, you can define an enforced mapping for this code point. For example, this can be used to delete this character if no code point list is provided or to translate this character always in another value. This can also be used to convert code points outside of a subset into this subset.

In contrast to the transliteration (+ is only done if a character doesn't exist in the target encoding), the mapping is always done for each character. Invalid (−) code points can be used to deactivate a code point. To activate (+) or deactivate (−) a range of code points you can use the minus mark (−) followed by one sub code point. The range goes from the smaller code point to the bigger one. For example:

```
(-00-7F) deactivate all US-ASCII code points
(+39-30) activate all decimal digits
```

The range operator (−) with the optional plus sign (+) can also be used to activate code points.

To define an own case mapping or folding (required for certain subsets) the caret sign (^) can be used. This mapping is only done if the user table is activated for case mapping (CASE=USRTAB).

To define transliterations/substitutions/mappings an assignment (=) of a code point list is required. The code point list can contain a maximum of 8 code points. If no code point list is specified, then the character is ignored. A transliteration to itself simply activates this code point. The same is valid for mapping definitions. A code point is a hexadecimal number representing a 21 bit UNICODE point.

```
00000000 to 001FFFFF hex
```

If a code point is not defined in the substitution table (USRTAB and/or SYSTAB), the appropriate substitution character (SUBCHR) is used. If SUBCHR is not set, no substitution is performed. Depending on the MODE, character conversion stops with an error (STOP) or ignores the character (IGNORE).

Text before and after brackets are comments.

Between "(" and "=" or "/" or ")" you can define hex digits until the first non-hex digit. All non-hex digits up to the next separator are interpreted as comment. Leading whitespace is ignored.

```
  REPLACE GERMAN SZ (00DF=0073/0073)                  WITH ss
THIS IS AN EXAMPLE FOR COMMENTS IN CODE POINT LIST:
  REPLACE EURO MARK (20AC= 45#E / 55#U / 52#R / 4F#O) WITH EURO
  REPLACE BOM MARK  (EFFF=)                           WITH NOTHING
```

Leading zeros are allowed. If you don't supply a hex value, then 0x00 is used.

```
  REPLACE GERMAN SZ (00DF=/)                   WITH 0x00 0x00
  REPLACE EURO MARK (20AC=00000045/00055/000052/000004F) WITH EURO
```

ATTENTION: Please don't use parentheses "()" or the operators in your comments.

To describe your own subsets, you can use a user table without a system table. The USRTAB can also be used to overwrite or add transliterations when a system substitution table (for example SYSTAB=ICONV) is used. The transliteration works recursively, that is if one of the substitution code points is not in the target set, this substitution will be used instead and so on.

```
  REPLACE GERMAN OE (0000D6=004F/0045)       WITH O E
  REPLACE GERMAN SZ (0000DF=0073/0073)       WITH s s
  REPLACE EURO MARK (0020AC=00D6/00DF/52/4F) WITH OE SZ R O
```

If you have a EURO sign in your text and convert it to *Latin1*, the resulting byte string will be *D6DF524F*. On the other hand, if you convert it to *ASCII*, the byte string will be *4F457373524F*.

The mapping as described above is always done and is also done recursive including the transliteration result. For example, if you define a mapping `(*30=39)` then the target data won't have any zero in the resulting text.

By replacing other code points recursively, you could easily cause infinite loops. To prevent this, the amount of replacements and the length of one replacement is limited to a maximum of 64. Note that this could still result in a large expansion of data. Therefore, be careful about defining recursive substitutions.

A sample user table for the ICONV system table to change the transliteration of German umlauts to AE, OE or UE is located in the SAMPLE directory under CCUTDEXL. Another sample user table called CCUTNPAS defines the *string.latin* subset (XOEF) which is mainly used for statutory reporting.

The order of definition is the order of processing the definitions. For exampel, if you define a mapping or a transliteration for code point X and later you make this code point invalid, then the code point is invalid. On the other hand, you can first deactivate all code points, then activate your subset and define your transliterations (best fit) or mappings.

For example: To define a non-expansion best fit mapping a user table can be used to delete all combined characters for single byte code pages. For the ICONV system transliteration table, a sample user table can be found under the name CCUTBFM1.

Some other subset user table definitions were added. For example:

- currently valid SEPA characters (<128) for money transfers (CCUTSEPA)
- characters valid in ISO-8859-15, CP1252 and IBM1142 (CCUTDELA)
- characters valid in ISO-8859-15, CP1252, IBM1142 and XOEF (CCUTDLAX)

All user table definitions are pre-calculated once at the beginning of execution to reduce the required CPU time for the conversion. So, the use of a user table increases the cost to open a file but has no effect on CPU utilization when converting the data.

To minimize the effort and to simplify the usage of user table, the most common subsets can be selected by keyword. In this case, a pre-calculated load module (DLL) is used. The subset definition can be for UTF (21 Bit UNICODE) or UCS (16 Bit UNICODE). If it is a UCS subset, then no valid code point is greater than 0xFFFF and you must use UCS CCSIDS to ensure that no code point greater than 0xFFFF is accepted.

If you have a working user table definition for a popular subset or system table manipulation, you can request a keyword for it. Please report this requirement over our issue tracking system

```
http://www.flam.de/en/technology/support/issues/
```

and attach the corresponding user table definition.

SELECTIONS

- `NPAS -UCS subset for statutory reporting (New passport, XOEF, String.Latin)`
- `SEPA -UCS subset of valid SEPA character smaller then 128`
- `DELA -UCS subset for German Latin of IBM1141, ISO8859-15 and CP1252`
- `DLAX -UCS subset for German Latin of IBM1141, ISO8859-15, CP1252 and XOEF`

4.2.208 PARAMETER *REPORTFILE*

SYNOPSIS

```
HELP:   Name of the report file to log substitutions
PATH:   CONV.WRITE.XML
TYPE:   STRING
SYNTAX: REPORTFILE='str'/STDOUT/STDERR
```

DESCRIPTION With the report capability, you can log every substitution (stop / ignore / substitute / translit / skip / ...) to a log file. For the report file, all rules and replacements are valid, which are described under "FILENAME HANDLING" above. For example:

```
REPORT=STDERR                          ; is SYSPRINT on z/OS
REPORT='~.MYREPORT(REPORT1)'           ; PDS on z/OS
REPORT='<SYSUID>.MYREPRT2'             ; PS dataset on z/OS
REPORT='~/myreport3.txt'               ; Unix path name
REPORT='<HOME>\reports\myreport4.txt'  ; Windows path name
```

A substitution report has the syntax listed below:

```
report_file         -> sub_definition_list
sub_definition_list -> sub_definition sub_definition_list
                    |  EMPTY
sub_definition      -> SUB '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    -> XLT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    |  IGN '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  STP '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  IDT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  SKP '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  UKW '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
code_point_list     -> CODEPOINT '/' code_point_list
                    |  EMPTY
code_point          -> CODEPOINT
                    |  EMPTY
byte_string         -> '(' BYTESTRING ')'
                    |  EMPTY
```

A comment behind brackets explains the reason for substitution:

```
'invalid character'
'incomplete character'
'incomplete character at the end of a record'
```

The notations STP (stop), IGN (ignore), SUB (substitute), XLT (transliteration), IDT (identical) or SKP (skip) before round brackets indicate the action taken. If the task can not be determined then UKW (unknown) is used. A ignore of an invalid character is realized as substitution with nothing. All substitutions with a length of 0 are indicated with IGN (ignore). The position of such an abnormal/irregular character is coded inside the braces:

- UNIT number of records or blocks (depends on the data)
- OFFSET byte offset up to this unit in the complete data stream
- POSITION number of bytes inside the unit where the activity was done

All counting starts with 0. The first block or record has unit 0.

The position is relative to the data stream which was presented to the character conversion module. After conversions and formatting the position values can be far off from the input file which was read originally. For example, when reading a text file with text formatting to records, then delimiting characters are not included in the byte offset, the units are records and the position indicates which character was the issue for this log entry.

The CODEPOINTs (UNICODE) are taken from the defined substitution table The BYTESTRINGs are taken from the input (read characters) and output data (written characters). BYTESTRINGs and CODEPOINTs are provided if they are known in the cases of such an irregularity. In some cases, these values can be missing.

For interpreting the report file, you must know that only the values inside round brackets are meaningful. Text before or behind round brackets are comments, but this may be subject to changes.

SELECTIONS

- `STDOUT -Write output to stdout`
- `STDERR -Write output to stderr`

4.2.209 OVERLAY *COMPRESS*

SYNOPSIS

```
HELP:   Compression procedure [NONE]
PATH:   CONV.WRITE.XML
TYPE:   OVERLAY
SYNTAX: COMPRESS.{GZIP()/BZIP2()/XZ()/FLAM4()}
```

DESCRIPTION With the overlay compress you can choose which kind of compression method is used for the data. Depending on the compression method different parameter can be provided to define the behavior in more detail.

If no compression method selected no compression of the data will be done.

See below the different possibilities of compression.

4.2.210 OBJECT *GZIP*

SYNOPSIS

```
HELP:   Compress data compliant to RCF1952 (GZIP)
PATH:   CONV.WRITE.XML.COMPRESS
TYPE:   OBJECT
SYNTAX: GZIP(LEVEL=num/COPY/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO, ↩
    COMMENT='str',CHECK)
```

DESCRIPTION The GZIP compression conforms to RCF1951/52. The compression level can be set to values from 0 to 9 (0=copy, 1=fastest, 9=best). Additionally, it is possible to set a comment and to enforce checksum calculation of the file header.

The checksum calculation is not recommended because only the newer versions of Zlib can handle this checksum technique. With older versions of Zlib this can cause problems. Therefore, checksum calculation is turned off by default.

The decompressor also accepts the older Zlib wrapping format specified in RFC1950. It is not possible to enforce RFC1950 at compression because most standards require RFC1952 for the deflate algorithms.

If no compression level is defined and the data is marked is being without redundancy, level 0 (copy) is used. In all other cases the default level 6 is used.

Decompression of concatenated GZIP files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
```

ARGUMENTS

- NUMBER:LEVEL=num/COPY/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level [AUTO==6]
  - COPY -No compression =copy of data (0)
  - FAST -Fastest compression (level 1)
  - LEV1 -Compression level 1
  - LEV2 -Compression level 2
  - LEV3 -Compression level 3
  - LEV4 -Compression level 4
  - LEV5 -Compression level 5
  - LEV6 -Compression level 6
  - LEV7 -Compression level 7
  - LEV8 -Compression level 8

  - LEV9 -Compression level 9
  - BEST -Best compression (level 9)
  - AUTO -Automatic compression (level 6)
- STRING:COMMENT='str' -Comment for header [NONE]
- SWITCH:CHECK -Calculate checksum over GZIP header [FALSE]

4.2.211 OBJECT *BZIP2*

SYNOPSIS

```
HELP:   BZIP2 data compression technique
PATH:   CONV.WRITE.XML.COMPRESS
TYPE:   OBJECT
SYNTAX: BZIP2(LEVEL=num/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO)
```

DESCRIPTION The BZIP2 algorithm compresses chunks of data. Such a chunk can be up to 100KB, 200KB, . . . , 900KB. The chunk size is selected automatically based on the internal block size (block size in KB rounded up to the next possible chunk size).

The chunk size directly correlates with the compression level. Compression levels range from 1 (fastest) to 9 (best, but slowest). Level 1 results in 100KB chunk size, level 9 in 900KB chunk size (i.e. chunk size = level*100KB). Hence, a lower level also reduces the required memory resources during compression and decompression.

Decompression of concatenated BZIP2 files is also possible. All concatenated files will be decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
```

ARGUMENTS

- NUMBER:LEVEL=num/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level (defines block size) [AUTO]
  - FAST -Fastest compression (Block size =100kb)
  - LEV1 -Compression level 1 (Block size =100kb)
  - LEV2 -Compression level 2 (Block size =200kb)
  - LEV3 -Compression level 3 (Block size =300kb)
  - LEV4 -Compression level 4 (Block size =400kb)
  - LEV5 -Compression level 5 (Block size =500kb)
  - LEV6 -Compression level 6 (Block size =600kb)
  - LEV7 -Compression level 7 (Block size =700kb)
  - LEV8 -Compression level 8 (Block size =800kb)
  - LEV9 -Compression level 9 (Block size =900kb)
  - BEST -Best compression (Block size =900kb)
  - AUTO -Automatic compression (depending on the real block size)

4.2.212 OBJECT *XZ*

SYNOPSIS

```
HELP:   XZ (LZMA) data compression technique
PATH:   CONV.WRITE.XML.COMPRESS
TYPE:   OBJECT
SYNTAX: XZ(LEVEL=num/FAST/LEV0/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO)
```

DESCRIPTION The XZ compression conforms to the XZ-compression tool on UNIX-systems. The compression level can be set to values from 0 to 9 (0=fastest, 9=best).

The default level is 6 when no level is specified.

Decompression of concatenated XZ files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
```

ARGUMENTS

- NUMBER:LEVEL=num/FAST/LEV0/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level [AUTO==6]
  - FAST -Fastest compression (level 0)
  - LEV0 -Compression level 0
  - LEV1 -Compression level 1
  - LEV2 -Compression level 2
  - LEV3 -Compression level 3
  - LEV4 -Compression level 4
  - LEV5 -Compression level 5
  - LEV6 -Compression level 6
  - LEV7 -Compression level 7
  - LEV8 -Compression level 8
  - LEV9 -Compression level 9
  - BEST -Best compression (level 9)
  - AUTO -Automatic compression (level 6)

4.2.213 OBJECT *FLAM4*

SYNOPSIS

```
HELP:   Compress data with FLAM
PATH:   CONV.WRITE.XML.COMPRESS
TYPE:   OBJECT
SYNTAX: FLAM4(MODE=CX7/CX8/VR8/ADC/NDC,COMMENT='str')
```

DESCRIPTION The FLAM4 compression to write compressed data into FLAMFILEs. As main argument you can define the compression mode (ADC, VR8, CX8). Additionally, it is possible to set a comment in the file header.

If the record-oriented method (write.record()) is used, the records with all attributes are written to the FLAM archive. If a block-oriented method (write.binary/char/text/xml()) is used, the data blocks are wrapped into records and written to the FLAMFILE as binary chunks with an undefined record format (U).

ARGUMENTS

- NUMBER:MODE=CX7/CX8/VR8/ADC/NDC -Compression mode [ADC]
  - CX7 -Compression encoding as printable characters
  - CX8 -Compression encoding as binary data
  - VR8 -Enhanced compression encoding as binary data
  - ADC -Advanced Data Compression
  - NDC -No data compression
- STRING:COMMENT='str' -Comment for the member in the FLAMFILE [internal use]

4.2.214 OVERLAY *ENCRYPT*

SYNOPSIS

```
HELP:   Encryption procedure [NONE]
PATH:   CONV.WRITE.XML
TYPE:   OVERLAY
SYNTAX: ENCRYPT/ENC.{F4PWD()/F4KME()}
```

DESCRIPTION The overlay *encrypt* allows you to choose which kind of encryption method is used for the data. Depending on the encryption method, different parameters can be set.

If no encryption method is selected, then no data encryption will be done.

See below for the encryption possibilities.

4.2.215 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password encryption
PATH:   CONV.WRITE.XML.ENCRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(MODE=AES/FLAM,PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Activates FLAM4 password-based encryption to encrypt the data written to FLAMFILEs. You can choose the encryption mode. The default and recommended mode is AES. If you provide no password, a hardcoded default password is used.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

If the data is already compressed, then MODE=NDC (no data compression) is used. Otherwise, ADC compression is used. If you set the compression mode to CX7/CX8/VR8 and enable encryption, e.g.

```
comp.flam(mode=CX7/CX8/VR8) encr.f4pwd()
```

the compression mode is ignored and ADC/NDC is used instead. Due to how those three old compression modes work, they are incompatible with FLAM encryption.

ARGUMENTS

- NUMBER:MODE=AES/FLAM -Encryption mode [AES]
  - AES -AES encryption
  - FLAM -FLAM encryption
- STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']
  - DEFAULT -FLAM4 default password

4.2.216 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   CONV.WRITE.XML.ENCRYPT
TYPE:   OBJECT
SYNTAX: F4KME(MODE=AES/FLAM,LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 encryption of data written to FLAMFILEs based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME parameter is provided (encr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using encr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

You may define the encryption mode. The default and recommended mode is AES.

If the data is already compressed, then MODE=NDC (no data compression) is used. Otherwise, ADC compression is used to create the FLAMFILE. If you set the compression mode to CX7/CX8/VR8 and enable encryption, e.g.

```
comp.flam(mode=CX7/CX8/VR8) encr.f4kme()
```

the compression mode is ignored and ADC/NDC is used instead. Due to how those three old compression modes work, they are incompatible with FLAM encryption.

ARGUMENTS

- NUMBER:MODE=AES/FLAM -Encryption mode [AES]
  - AES -AES encryption

  - FLAM -FLAM encryption

- STRING:LIBRARY='str' -Library name for FKME ['libfkme']
- STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']
- STRING:PARAMETER='str' -Parameter for the call of FKME [”]

4.2.217 OVERLAY *ENCODE*

SYNOPSIS

```
HELP:   Encoding procedure [NONE]
PATH:   CONV.WRITE.XML
TYPE:   OVERLAY
SYNTAX: ENCODE/COD.{BASE64()/BASE32()/BASE16()/OPGP()}
```

DESCRIPTION The overlay *encode* allows to choose which kind of encoding method is used for the data. Depending on the encoding method, different parameters can be set.

If no encoding method is selected, then no data encoding will be done.

See below for the available encoding options.

4.2.218 OBJECT *BASE64*

SYNOPSIS

```
HELP:   Base64 encoding (RFC4648)
PATH:   CONV.WRITE.XML.ENCODE
TYPE:   OBJECT
SYNTAX: BASE64(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)

  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.219 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.XML.ENCODE.BASE64
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.220 OBJECT *BASE32*

SYNOPSIS

```
HELP:   Base32 encoding (RFC4648)
PATH:   CONV.WRITE.XML.ENCODE
TYPE:   OBJECT
SYNTAX: BASE32(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- `NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]`
  - `NONE -No character set defined`
  - `SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)`
  - `ASCII -ASCII (mainly used in the for open system)`
  - `UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)`
  - `UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)`
  - `EBCDIC -EBCDIC (mainly used on IBM mainframe)`
  - `UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)`
  - `UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)`
  - `UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)`
  - `UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)`
  - `UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)`
  - `UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)`
  - `UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)`
  - `UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)`
- `NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]`
  - `BLOCK -Block-oriented conversion (all data is one chunk)`

  - RECORD -Record-oriented conversion (convert each record individually)

- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.221 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.XML.ENCODE.BASE32
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8
```

```
PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHA+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHA+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.222 OBJECT *BASE16*

SYNOPSIS

```
HELP:   Base16 encoding (RFC4648)
PATH:   CONV.WRITE.XML.ENCODE
TYPE:   OBJECT
SYNTAX: BASE16(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)

  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.223 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.XML.ENCODE.BASE16
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.224 OBJECT *OPGP*

SYNOPSIS

```
HELP:   Base64 encoding with 76 character, CRC checksum and Armor clasp (RFC4880)
PATH:   CONV.WRITE.XML.ENCODE
TYPE:   OBJECT
SYNTAX: OPGP(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE ↩
    /UTF32BE/UCS4LE/UTF32LE,DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM)
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)

4.2.225 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for write operation
PATH:   CONV.WRITE.XML
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.2.226 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for write operation
PATH:   CONV.WRITE.XML
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.2.227 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of output data
PATH:   CONV.WRITE.XML
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.2.228 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   CONV.WRITE.XML.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- `STRING:VALUE='str' -Checksum as hex-string`
- `STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`

4.2.229 OBJECT *RECORD*

SYNOPSIS

```
HELP:   Write records to a file
PATH:   CONV.WRITE
TYPE:   OBJECT
SYNTAX: RECORD(FILE='str'/STREAM/DUMMY,FALLOC(),REMAIN,APPEND,FLUSH,RECMODE=STOP/CUT/WRAP, ↩
    RECDELIM='bin'/CR-ASCII/LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF-EBCDIC/NL-EBCDIC/CRLF- ↩
    EBCDIC/CR-UTF08/LF-UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE/NL-UTF16BE/CRLF- ↩
    UTF16BE/CR-UTF16LE/LF-UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF-UTF32BE/NL-UTF32BE/ ↩
    CRLF-UTF32BE/CR-UTF32LE/LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE,LENFORMAT.{},PRNCONTROL= ↩
    DETACH/RETAIN/ERASE,SUPPADDING,PADCHAR='bin'/BINARY-ZERO/ASCII-BLANK/EBCDIC-BLANK/UTF08- ↩
    BLANK/UTF16BE-BLANK/UTF16LE-BLANK/UTF32BE-BLANK/UTF32LE-BLANK,CCSID='str'/DEFAULT/ASCII/ ↩
    EBCDIC,CASE=UPPER/LOWER/FOLD/SUPPER/SLOWER/USRTAB,BOM,CHRMODE=STOP/IGNORE/SUBSTITUTE, ↩
    SUBCHAR[num/SYSTEM...],SYSTABLE=ICONV,USRTABLE='str'/NPAS/SEPA/DELA/DLAX,REPORTFILE='str ↩
    '/STDOUT/STDERR,COMPRESS/CMP.{},ENCRYPT/ENC.{},ENCODE/COD.{},LANG='str',PLATFORM=WIN/UNX ↩
    /ZOS/USS/VSE/BS2/MAC,HASH(),INDSIZ=num,INDCHR=SPACE/TABULATOR,NOCMNT)
```

DESCRIPTION Write record formats the list of elements as records for host data set formats.

The following options are supported:

- Cutting or wrapping of records which are too long

- ASA and machine print control character handling
- All kind of mainframe records formats, length and block size
- Different record formats for the distributed world
- Free definition of padding character for fix length formats

If a record is marked as text record, character conversion from UTF-8 to a defined CCSID is done.

The following options are supported for character conversion:

- Stop, ignore or substitution of invalid or incomplete character
- A free defined substitution character list
- Different system substitution / transliteration tables
- A user defined substitution table to change the system tables
- Case management (upper, lower, folding and special casing)
- Transliteration between subsets (mapping) and normalization
- BOM management and automatic detection of char sets
- Reporting of ignored or substituted characters

At record I/O, only the encryption and compression with FLAM are supported. Other popular compression methods like GZIP lose the record boundaries. In this case, for text records please use *write.text()* together with one of the available methods to add a delimiter at the and of each record to maintain the record boundaries.

Write record supports different encoding schemes (e.g. Base64). By default, the encoding is done record by record. If you enforce block orientation for the encoding, the record boundaries are lost and the resulting blocks can wrapped to new records, if you define all the corresponding parameters accordingly.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to write ["==origin.ext]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`
- `SWITCH:REMAIN -Remain existing files (don't overwrite) [FALSE]`
- `SWITCH:APPEND -Append data to this file [FALSE]`
- `SWITCH:FLUSH -Enforce flush in front of close file [FALSE]`
- `NUMBER:RECMODE=STOP/CUT/WRAP -Mode used to write records [STOP]`
  - `STOP -Stop if record too long`
  - `CUT -Cut if record too long`
  - `WRAP -Wrap if record too long`
- `STRING:RECDELIM='bin'/CR-ASCII/LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF-EBCDIC/NL-EBCDIC/CRLF-EBCDIC/CR-UTF08/LF-UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE/NL-UTF16BE/CRLF-UTF16BE/CR-UTF16LE/LF-UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF-UTF32BE/NL-UTF32BE/CRLF-UTF32BE/CR-UTF32LE/LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE -Delimiter used to define end of record`
  - `CR-ASCII -Delimiter:carriage return in ASCII (0x0D)`
  - `LF-ASCII -Delimiter:line feed in ASCII (0x0A)`

  - NL-ASCII -Delimiter:new line in ASCII (0x85)
  - CRLF-ASCII -Delimiter:carriage return line feed in ASCII (0x0D0A)
  - CR-EBCDIC -Delimiter:carriage return in EBCDIC (0x0D)
  - LF-EBCDIC -Delimiter:line feed in EBCDIC (0x25)
  - NL-EBCDIC -Delimiter:new line in EBCDIC (0x15)
  - CRLF-EBCDIC -Delimiter:carriage return line feed in EBCDIC (0x0D25)
  - CR-UTF08 -Delimiter:carriage return in UTF-8 (0x0D)
  - LF-UTF08 -Delimiter:line feed in UTF-8 (0x0A)
  - NL-UTF08 -Delimiter:new line in UTF-8 (0xC285)
  - CRLF-UTF08 -Delimiter:carriage return line feed in UTF-8 (0x0D0A)
  - CR-UTF16BE -Delimiter:carriage return in UTF-16BE (0x000D)
  - LF-UTF16BE -Delimiter:line feed in UTF-16BE (0x000A)
  - NL-UTF16BE -Delimiter:new line in UTF-16BE (0x0085)
  - CRLF-UTF16BE -Delimiter:carriage return line feed in UTF-16BE (0x000D000A)
  - CR-UTF16LE -Delimiter:carriage return in UTF-16LE (0x0D00)
  - LF-UTF16LE -Delimiter:line feed in UTF-16LE (0x0A00)
  - NL-UTF16LE -Delimiter:new line in UTF-16LE (0x8500)
  - CRLF-UTF16LE -Delimiter:carriage return line feed in UTF-16LE (0x0D000A00)
  - CR-UTF32BE -Delimiter:carriage return in UTF-32BE (0x0000000D)
  - LF-UTF32BE -Delimiter:line feed in UTF-32BE (0x0000000A)
  - NL-UTF32BE -Delimiter:new line in UTF-32BE (0x00000085)
  - CRLF-UTF32BE -Delimiter:carriage return line feed in UTF-32BE (0x0000000D0000000A)
  - CR-UTF32LE -Delimiter:carriage return in UTF-32LE (0x0D000000)
  - LF-UTF32LE -Delimiter:line feed in UTF-32LE (0x0A000000)
  - NL-UTF32LE -Delimiter:new line in UTF-32LE (0x85000000)
  - CRLF-UTF32LE -Delimiter:carriage return line feed in UTF-32LE (0x0D0000000A000000)
- SWITCH:SUPPADDING -Suppress trailing padding characters (useful for variable records)
- STRING:PADCHAR='bin'/BINARY-ZERO/ASCII-BLANK/EBCDIC-BLANK/UTF08-BLANK/UTF16BE-BLANK/UTF16LE-BLANK/UTF32BE-BLANK/UTF32LE-BLANK -Character to pad records to a fixed length [system]
  - BINARY-ZERO -Padding with 0x00
  - ASCII-BLANK -Padding with 0x20
  - EBCDIC-BLANK -Padding with 0x40
  - UTF08-BLANK -Padding with 0x20
  - UTF16BE-BLANK -Padding with 0x0020
  - UTF16LE-BLANK -Padding with 0x2000
  - UTF32BE-BLANK -Padding with 0x00000020
  - UTF32LE-BLANK -Padding with 0x20000000
- NUMBER:INDSIZ=num -Number of XML indentation characters per indentation level [2 (SPACE) or 1 (TAB)]
- NUMBER:INDCHR=SPACE/TABULATOR -Indentation character used to indent XML tags [SPACE]
  - SPACE -Use the space character as indentation character
  - TABULATOR -Use the tabulator character as indentation character
- SWITCH:NOCMNT -Do not write out XML comments [FALSE]

4.2.230 OBJECT *FALLOC*

SYNOPSIS

```
HELP:   Platform dependent file allocation parameter
PATH:   CONV.WRITE.RECORD
TYPE:   OBJECT
SYNTAX: FALLOC(ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZE=num,RECFORMAT=NONE/BIN/TXT/ ↩
    DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX- ↩
    REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR ↩
    /FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/ ↩
    VRA/VRM,RECLENGTH=num,KEYDISP=OLD/NEW/DEL,KEYPOSITION=num,KEYLENGTH=num,SUBSYSTEM(), ↩
    SPACE(),LIKE='str',RENEW)
```

DESCRIPTION The file allocation object collects several parameters for platforms which are record oriented based on a catalog system like z/OS® and where the space, volume and other things must be predefined. The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

But also on distributed platforms like (WIN, UNIX, MAC, ...) FLAM supports record oriented file types to keep the record information on such platforms. For this case the different record format for the open world can be defined. To read such a special file format on an open platform the correct record format must also be defined. On a catalog based system, this is not required. If the record format in the catalog are not the same as specified an error will occur.

ARGUMENTS

- NUMBER:ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [SEQ]
  - SEQ -Sequential data set [DEFAULT]
  - LIB -Library (ZOS-PDSE)
  - PDS -Partitioned data set (ZOS-PDS)
  - USS -UNIX files (path based)
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)
- NUMBER:BLKSIZE=num -Block size [system]
- NUMBER:RECFORMAT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format [auto]
  - NONE -No format defined
  - BIN -Open -Binary (undefined) data (only the last record are shorter then record length)
  - TXT -Open -Text data with standard delimiter (result is a stripped variable length records)
  - DLM -Open -Records separated by a special delimiter with the result of variable length records
  - VAR -Open -Variable length record (if necessary define length format)
  - VAR-ASA -Open -Variable length record with ASA print control character (if necessary define length format)
  - VAR-MCC -Open -Variable length record with machine print control codes (if necessary define length format)
  - FIX -Open -Fix length record (file size =N * record length)

- FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
- FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
- VAR-REL -Open -Variable relative record (if necessary define length format)
- VAR-REL-ASA -Open -Variable relative record with ASA print control character (if ne cessary define length format)
- VAR-REL-MCC -Open -Variable relative record with machine print control codes (if ne cessary define length format)
- FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
- FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size = N * (record length + 9))
- FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size = N * (record length + 9))
- F -Host -Fixed-length, unblocked
- FA -Host -Fixed-length, unblocked, ASA print control characters
- FM -Host -Fixed-length, unblocked, machine print control codes
- FS -Host -Fixed-length, standard
- FSA -Host -Fixed-length, standard, ASA print control characters
- FSM -Host -Fixed-length, standard, machine print control codes
- FB -Host -Fixed-length, blocked
- FBA -Host -Fixed-length, blocked, ASA print control characters
- FBM -Host -Fixed-length, blocked, machine print control codes
- FBS -Host -Fixed-length, blocked, standard
- FBSA -Host -Fixed-length, blocked, standard, ASA print control characters
- FBSM -Host -Fixed-length, blocked, standard, machine print control codes
- FE -Host -Fixed-length, ESDS
- FEA -Host -Fixed-length, ESDS, ASA print control characters
- FEM -Host -Fixed-length, ESDS, machine print control codes
- FK -Host -Fixed-length, KSDS
- FKA -Host -Fixed-length, KSDS, ASA print control characters
- FKM -Host -Fixed-length, KSDS, machine print control codes
- FR -Host -Fixed-length, RRDS
- FRA -Host -Fixed-length, RRDS, ASA print control characters
- FRM -Host -Fixed-length, RRDS, machine print control codes
- U -Host -Undefined-length
- UA -Host -Undefined-length, ASA print control characters
- UM -Host -Undefined-length, machine print control codes
- V -Host -Variable
- VA -Host -Variable, ASA print control characters
- VM -Host -Variable, machine print control codes
- VS -Host -Variable, spanned
- VSA -Host -Variable, spanned, ASA print control characters
- VSM -Host -Variable, spanned, machine print control codes
- VB -Host -Variable, blocked

  - VBA -Host -Variable, blocked, ASA print control characters
  - VBM -Host -Variable, blocked, machine print control codes
  - VBS -Host -Variable, blocked, spanned
  - VBSA -Host -Variable, blocked, spanned, ASA print control characters
  - VBSM -Host -Variable, blocked, spanned, machine print control codes
  - VE -Host -Variable, ESDS
  - VEA -Host -Variable, ESDS, ASA print control characters
  - VEM -Host -Variable, ESDS, machine print control codes
  - VK -Host -Variable, KSDS
  - VKA -Host -Variable, KSDS, ASA print control characters
  - VKM -Host -Variable, KSDS, machine print control codes
  - VR -Host -Variable, RRDS
  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes
- NUMBER:RECLENGTH=num -Record length (data only, no print control and length fields)
- NUMBER:KEYDISP=OLD/NEW/DEL -Key disposition for index/key sequential data sets [OLD]
  - OLD -Key disposition OLD (keep the existing key)
  - NEW -Key disposition NEW (insert record number as key)
  - DEL -Key disposition DEL (delete key from record)
- NUMBER:KEYPOSITION=num -Key position (>=1) for index/key sequential data sets [1]
- NUMBER:KEYLENGTH=num -Key length (>=1) for index/key sequential data sets [8]
- STRING:LIKE='str' -Specifies the name of the model data set from which the attributes are to be copied
- SWITCH:RENEW -Enforce new allocation by a remove of an existing file

4.2.231 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   CONV.WRITE.RECORD.FALLOC
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
SUBSYS(NAME='name' parm='keyword,keyword=value,value')
SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- STRING:NAME='str' -Name of the subsystem
- STRING:PARM='str' -Parameter list for the subsystem (comma separated)

4.2.232 OBJECT *SPACE*

SYNOPSIS

```
HELP:   Platform dependent space definition parameter
PATH:   CONV.WRITE.RECORD.FALLOC
TYPE:   OBJECT
SYNTAX: SPACE(PRIMARY=num,SECONDARY=num,DIRBLOCKS=num,VOLSER='str',VOLUNIT='str',DATACLASS ↩
    ='str',MGMTCLASS='str',STORCLASS='str',NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG, ↩
    ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG,RELEASE=YES/NO,UNMOVABLE,CONTIG,PERMANENT,RLS= ↩
    NRI/CR/CRE)
```

DESCRIPTION The space definition object collects several parameters for platforms where data sets are preallocated like z/OS® and where clylinders or tracks must be predefined for a file allocation.

The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

All size values must be given in megabyte (MiB). The calculation in the platform dependent units (tracks, cylinders, ...) are done inernally. Note: 1 MiB = 1024 KiB = 1024*1024 Byte

ARGUMENTS

- NUMBER:PRIMARY=num -Primary size in megabytes
- NUMBER:SECONDARY=num -Secondary size in megabytes
- NUMBER:DIRBLOCKS=num -Directory blocks for partitioned data sets
- STRING:VOLSER='str' -Volume serial number
- STRING:VOLUNIT='str' -Volume unit
- STRING:DATACLASS='str' -Data class
- STRING:MGMTCLASS='str' -Management class
- STRING:STORCLASS='str' -Storage class
- NUMBER:NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for normal end [CATALOG]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for abnormal end [DELETE]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete

  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:RELEASE=YES/NO -Release unused space when file is closed [SEQ=YES, PDS=NO]
  - YES -Yes, do it
  - NO -No, don't do it
- SWITCH:UNMOVABLE -Allocate space unmovable [FALSE]
- SWITCH:CONTIG -Allocate space contiguously [FALSE]
- SWITCH:PERMANENT -Allocate space permanently [FALSE]
- NUMBER:RLS=NRI/CR/CRE -Specifies the type of record level sharing (RLS) being done for a specific data set [NONE]
  - NRI -No Read Integrity
  - CR -Consistent Read
  - CRE -Consistent Read Explicit

4.2.233 OVERLAY *LENFORMAT*

SYNOPSIS

```
HELP:   Format of the length field for open platforms [word]
PATH:   CONV.WRITE.RECORD
TYPE:   OVERLAY
SYNTAX: LENFORMAT.{INTEGER()/STRING()/BCD()/HOST()}
```

DESCRIPTION The overlay can be used to define the format of the length filed on open platforms which don't support record formats by the data management system. For example: If you read records on a mainframe and you want to write the records as records (not as binary or text data) on WINDOWS or UNIX then the data will be prefixed with the record length. The format of this length field can be changed over this union.

This length specification will be only used for variable length records. The default mapping method between record formats will be map each host record format the corresponding open variable format.

If the length format not defined the length field will be written as 32 bit integer in platform dependent byte order. This behavior corresponds to the FILEDATA=RECORD parameter for USS files on z/OS. Means if you write a record oriented file to USS without any format changed over this overlay then you can read the record with another MVS application using FILEDATA=RECORD.

If you choose record format text or binary then this will correspond to FILEDATA=TEXT or FILEDATA=BINARY and the length format specification will be ignored. To enable record format mapping based on the data type you can set the environment variable below:

```
FL_RECORD_FORMAT_MAPPING=DATATYPE
```

If this environment variable set, then the length format is only used if you explicitly set a variable record format of the open systems.

Since version 5.1.8, FLAM can detect the 4 byte length fields (integer or host) inside a data stream. We recommend to use one of these formats to have more comfort when reading.

4.2.234 OBJECT *INTEGER*

SYNOPSIS

```
HELP:   Use integer format for the length field
PATH:   CONV.WRITE.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: INTEGER(ENDIAN=SYSTEM/BIG/LITTLE,WIDTH=U16/U32/U64,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in integer format. You can define the byte order, the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [SYSTEM]
  - SYSTEM -Byte order of the current system
  - BIG -Big endian byte order
  - LITTLE -Little endian byte order
- NUMBER:WIDTH=U16/U32/U64 -Bit width/rate [B32]
  - U16 -16 bit unsigned integer
  - U32 -32 bit unsigned integer
  - U64 -64 bit unsigned integer
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.2.235 OBJECT *STRING*

SYNOPSIS

```
HELP:   Use string format for the length field
PATH:   CONV.WRITE.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: STRING(CHRSET=SYSTEM/ASCII/EBCDIC,COUNT=C04/C05/C06,BASE=B10,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in string format (a character per digit). You can define the character set, the character count, the base and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:CHRSET=SYSTEM/ASCII/EBCDIC -Character set [SYSTEM]
  - SYSTEM -Use system code page
  - ASCII -Use ASCII (UTF-8) code page
  - EBCDIC -Use EBCDIC code page
- NUMBER:COUNT=C04/C05/C06 -Character count [C06]
  - C04 -4 character/digits
  - C05 -5 character/digits
  - C06 -6 character/digits
- NUMBER:BASE=B10 -String base [B10]
  - B10 -Base 10 (decimal)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.2.236 OBJECT *BCD*

SYNOPSIS

```
HELP:   Use BCD format for the length field
PATH:   CONV.WRITE.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: BCD(WIDTH=U16/U24/U32,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in BCD format (4 Bit are one decimal digit). You can define the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:WIDTH=U16/U24/U32 -Bit width/rate [B32]
  - U16 -16 bit unsigned BCD (POV with 4 digits)
  - U24 -24 bit unsigned BCD (POV with 6 digits)
  - U32 -32 bit unsigned BCD (POV with 8 digits)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.2.237 OBJECT *HOST*

SYNOPSIS

```
HELP:   Use host format (LLxx) for the length field
PATH:   CONV.WRITE.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: HOST(ENDIAN=SYSTEM/BIG/LITTLE,EXCLUDE)
```

DESCRIPTION The object defines the 4 byte host specific format (LLxx) of the length field. This format uses the first 16 bits as integer in big endian or little endian byte order to represent the record length including or excluding the record length field itself and the last 16 bits are set to zero when writing and ignored when reading.

For example, a record with 136 bytes content has a default length field (big endian, inclusive) of:

```
x'008D0000'
```

The default conforms to the length fields used by the data management system of MVS.

ARGUMENTS

- NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [BIG ENDIAN] for first 16 Bit
  - SYSTEM -Byte order of the current system
  - BIG -Big endian byte order
  - LITTLE -Little endian byte order
- SWITCH:EXCLUDE -Record length field not part of the length value [OFF]

4.2.238 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   CONV.WRITE.RECORD
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control character handling is a special feature for record oriented files of mainframe system. For print control, the ASA or machine control charaters are encoded in the first byte of the record.

If an attribute with such a control character is known on write operation, you can decide whether it will be part of the output (RETAIN), written to the output if required (DETACH) or never written (ERASE).

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.2.239 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion to this CCSID [DEFAULT]
PATH:   CONV.WRITE.RECORD
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC
```

DESCRIPTION The TO code defines the encoding string or CCSID for the output text stream. The default value of this CCSID for read operations is UTF-8. For write operations, the default value is derived from the environment variable LANG. If LANG is not set, the default ASCII or EBCDIC system-codepage is used, depending on the platform.

To get a list of supported encoding strings, please use the command below:

```
   flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs please use:

```
   flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
   CCSID='37'    will be equal to CCSID='IBM037'
   CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit. Optionally, you can use one of the keywords below to use the system-specific default ASCII or EBCDIC CCSID.

SELECTIONS

- `DEFAULT -Use default CCSID (system-specific)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`

4.2.240 PARAMETER *CASE*

SYNOPSIS

```
HELP:   Define upper, lower or special case conversion [NONE]
PATH:   CONV.WRITE.RECORD
TYPE:   NUMBER
SYNTAX: CASE=UPPER/LOWER/FOLD/SUPPER/SLOWER/USRTAB
```

DESCRIPTION With the case modifier, you can perform conversion to upper or lower case as well as apply case folding and some special case conversions.

Case folding converts all characters to lower case but is more compatible with language specific rules for doing so. For example, this may be useful for case-insensitive string comparison if the strings contain non-Latin characters (code point < 128).

This mapping is done for each code point in front of substitution or transliteration. The respective case is accomplished on the basis of a separate Unicode conversion table in memory. This conversion can be manipulated if user mapping is activated and mapping definitions ($\star$) are defined through the provided user table.

The options UPPER and LOWER convert characters without expansion, i.e. returned strings will never grow in length. Note, however, that case folding and the special options (convert e.g. ß to SS) can result in an expansion of data.

All these conversions are pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `UPPER –Change character to upper case`
- `LOWER –Change character to lower case`
- `FOLD –Change character to folded case (for compare)`
- `SUPPER –Change character to special upper case`
- `SLOWER –Change character to special lower case`
- `USRTAB –Change character according to user table`

4.2.241 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Write byte order mark in front of each record [FALSE]
PATH:   CONV.WRITE.RECORD
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the write byte order mark keyword is defined, then the corresponding BOM sign is written at the start of the output stream (block oriented) or at the beginning of each record. If this BOM keyword is not used, then the BOM sign will not be written (this is valid independent of the chosen encoding/CCSID). If a valid BOM sign is located in the middle of the input stream, it is removed from the data. If you define the BOM keyword and the target charset contains only single bytes or no valid BOM sign, then no byte order mark will be written to the output file.

If an invalid BOM sign is recognized, then it will be ignored and causes no entry in the report file. If the byte order changes in the middle of text and the BOM handling switch is not activated, the conversion will stop and an error message is thrown.

By default, the write BOM keyword is set to OFF.

4.2.242 PARAMETER *CHRMODE*

SYNOPSIS

```
HELP:   Define handling of non-convertible characters [STOP]
PATH:   CONV.WRITE.RECORD
TYPE:   NUMBER
SYNTAX: CHRMODE=STOP/IGNORE/SUBSTITUTE
```

DESCRIPTION With the MODE selection you can define how an invalid or incomplete character is handled inside of a unit by this component. All possible modes are listed at the end of this chapter.

This MODE is valid for the content of each unit (block or record). At the end of a unit there can be an incomplete character. Such a rest will be handled dependent on the METHOD you choose. The rest of the last unit of a data field will be concatenated to the first unit of the next data field. For all other records in a data field the handling of such a rest will be defined by the respective METHOD.

The SKIP case for METHOD is only relevant for incomplete byte sequences at the end of a unit. The ignore and substitution mode is only effective for all characters in the middle of the unit, besides an incomplete byte sequence occurs in the last unit. Please don't mix up the IGNORE case for MODE with the SKIP case of METHOD.

If a system or user substitution table is used the stop, ignore or substitute are only relevant for characters not managed by the CCSIDs and the substitution tables. This means that the USRTAB and SYSTAB are added to the capabilities of character conversion and only for the rest the chosen MODE is used.

4.2.242.1 CONSTANT *STOP*

SYNOPSIS

```
HELP:   Stop at the first non-convertible character
PATH:   CONV.WRITE.RECORD.CHRMODE
TYPE:   NUMBER
SYNTAX: STOP
```

DESCRIPTION Stops conversion at the first incomplete or invalid character which can not be converted. An error message is given with the position of abortion and the value in the input stream. An entry will be (STP()) written to the report file to log this event. To be compatible with other conversion tools, this is the default case, but we recommend to use IGNORE or SUBSTITUTE to prevent any abortion.

indexterm:[Constant STOP}

4.2.242.2 CONSTANT *IGNORE*

SYNOPSIS

```
HELP:   Ignore non-convertible characters
PATH:   CONV.WRITE.RECORD.CHRMODE
TYPE:   NUMBER
SYNTAX: IGNORE
```

DESCRIPTION Ignores incomplete or invalid characters and writes a report entry for each character. IGNORE is a substitution by no corresponding character that means no substitution table is required.

indexterm:[Constant IGNORE}

4.2.242.3 CONSTANT *SUBSTITUTE*

SYNOPSIS

```
HELP:   Substitute non-convertible characters
PATH:   CONV.WRITE.RECORD.CHRMODE
TYPE:   NUMBER
SYNTAX: SUBSTITUTE
```

DESCRIPTION Substitutes incomplete or invalid character and writes a report entry for each character. For SUBSTITUTE you can define a substitution character list (SUBCHR). If no SUBCHR is provided then this option substitutes invalid or incomplete characters with code point 0x00001A. If a SUBCHR is defined all invalid or incomplete characters will be substituted by the defined list of code points that can be up to 8. If you specify a SUBCHR the substitution mode will be enabled.

indexterm:[Constant SUBSTITUTE}

4.2.243 PARAMETER *SUBCHAR*

SYNOPSIS

```
HELP:   Default substitution character list [0x1A] (maximal 8 code points)
PATH:   CONV.WRITE.RECORD
TYPE:   NUMBER
SYNTAX: SUBCHAR[num/SYSTEM...]
```

DESCRIPTION As substitution character (SUBCHR) you can define a list with up to 8 UNICODE points which are used for the substitution of invalid or incomplete characters if the corresponding code point is not defined in the substitution table (see SUBTAB). If you assign to it the keyword SYSTEM the value 0x1A is written as default substitution character.

If you activate substitution and no SUBCHR is defined, then unsupported characters will be replaced by CODEPOINT 0x00001A.

On the other hand SUBCHR will replace every undefined code point in the substitution table, what is the same behavior when using a USRTAB with corresponding SUBCHR values for every possible code point (000000-200000). If SUBCHR but no MODE is defined the substitution mode will be activated.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `SYSTEM -Use system specific substitution char (0x1A for ASCII/UTF, 0x3F for EBCDIC)`

4.2.244 PARAMETER *SYSTABLE*

SYNOPSIS

```
HELP:   Define a preloaded system substitution table
PATH:   CONV.WRITE.RECORD
TYPE:   NUMBER
SYNTAX: SYSTABLE=ICONV
```

DESCRIPTION With the system table (SYSTAB) you can load a substitution table with a predefined transliteration schema. Entries in the USRTAB can be used to override predefined transliteration entries or add missing transliteration code points. The options listed at the end of this chapter are available.

If a substitution table is used then the MODE option (STOP/IGNORE/SUBSTITUTE) is only relevant for characters that cannot be converted or substituted.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `ICONV -Preload with ICONV//TRANSLIT substitution table`

4.2.245 PARAMETER *USRTABLE*

SYNOPSIS

```
HELP:   Name of the file containing the user conversion table
PATH:   CONV.WRITE.RECORD
TYPE:   STRING
SYNTAX: USRTABLE='str'/NPAS/SEPA/DELA/DLAX
```

DESCRIPTION With this parameter, you can choose a predefined subset or standard manipulation for the system tables via keyword (see `flcl help "xcnv.inp .sav.fil.cnv.chr.usrtab"` for available keywords). Alternatively, you can use a file containing a user defined substitution table (USRTAB).

For the user table file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
USRTAB=NPAS                          ; use XOEV predefined subset
USRTAB=DELA                          ; German Latin single byte subset
USRTAB='~.MYUSRTAB(USRTAB1)'         ; PDS on z/OS
USRTAB='<SYSUID>.MYUSRT02'           ; PS dataset on z/OS
USRTAB='~/myusrtab3.txt'             ; Unix path name
USRTAB='<HOME>\usrtab\myusrtab4.txt' ; Windows path name
```

A user defined substitution table has the syntax listed below:

```
usr_tab_file         -> usr_definition_list
usr_definition_list  -> usr_definition usr_definition_list
                     |  EMPTY
usr_definition       -> '(' codepoint_definition ')'
codepoint_definition -> cp_activation
                     |  cp_deactivation
                     |  cp_transliteration
                     |  cp_standard_mapping
                     |  cp_usr_case_mapping
cp_activation        -> ['+'] CODEPOINT  [ '-' CODEPOINT ]
cp_deactivation      ->  '-'  CODEPOINT  [ '-' CODEPOINT ]
cp_transliteration   -> ['+'] CODEPOINT '=' code_point_list
cp_standard_mapping  ->  '*'  CODEPOINT '=' code_point_list
cp_usr_case_mapping  ->  '^'  CODEPOINT '=' code_point_list
code_point_list      -> CODEPOINT separator code_point_list
                     |  CODEPOINT
                     |  EMPTY
separator            -> '/'
                     |  ','
                     |  ';'
```

Note: In the description below "/" is used as code point separator.

The optional signs in front of a code point (CP) have the following meaning:

```
'+' - a valid transliteration (only if CP not in the target set)
'*' - a valid mapping         (this translation is always done)
'^' - a case mapping/folding  (only if case=usrtab is activated)
'-' - an invalid definition   (removes CP for subset definitions)
```

If no sign is used, then a transliteration for a valid code point is defined. Valid code points are accepted, invalid code points result in in an error. Error handling depends on the mode (STOP,IGNORE, SUBSTITUTE). Definition of + code points without code point list, thus without = is equal definition of valid code points. With an asterisk $*$, you can define an enforced mapping for this code point. For example, this can be used to delete this character if no code point list is provided or to translate this character always in another value. This can also be used to convert code points outside of a subset into this subset.

In contrast to the transliteration (+ is only done if a character doesn't exist in the target encoding), the mapping is always done for each character. Invalid (−) code points can be used to deactivate a code point. To activate (+) or deactivate (−) a range of code points you can use the minus mark (−) followed by one sub code point. The range goes from the smaller code point to the bigger one. For example:

```
(-00-7F) deactivate all US-ASCII code points
(+39-30) activate all decimal digits
```

The range operator (−) with the optional plus sign (+) can also be used to activate code points.

To define an own case mapping or folding (required for certain subsets) the caret sign (^) can be used. This mapping is only done if the user table is activated for case mapping (CASE=USRTAB).

To define transliterations/substitutions/mappings an assignment (=) of a code point list is required. The code point list can contain a maximum of 8 code points. If no code point list is specified, then the character is ignored. A transliteration to itself simply activates this code point. The same is valid for mapping definitions. A code point is a hexadecimal number representing a 21 bit UNICODE point.

```
  00000000 to 001FFFFF hex
```

If a code point is not defined in the substitution table (USRTAB and/or SYSTAB), the appropriate substitution character (SUBCHR) is used. If SUBCHR is not set, no substitution is performed. Depending on the MODE, character conversion stops with an error (STOP) or ignores the character (IGNORE).

Text before and after brackets are comments.

Between "(" and "=" or "/" or ")" you can define hex digits until the first non-hex digit. All non-hex digits up to the next separator are interpreted as comment. Leading whitespace is ignored.

```
  REPLACE GERMAN SZ (00DF=0073/0073)                  WITH ss
THIS IS AN EXAMPLE FOR COMMENTS IN CODE POINT LIST:
  REPLACE EURO MARK (20AC= 45#E / 55#U / 52#R / 4F#O) WITH EURO
  REPLACE BOM MARK  (EFFF=)                           WITH NOTHING
```

Leading zeros are allowed. If you don't supply a hex value, then 0x00 is used.

```
  REPLACE GERMAN SZ (00DF=/)                   WITH 0x00 0x00
  REPLACE EURO MARK (20AC=00000045/00055/000052/000004F) WITH EURO
```

ATTENTION: Please don't use parentheses "()" or the operators in your comments.

To describe your own subsets, you can use a user table without a system table. The USRTAB can also be used to overwrite or add transliterations when a system substitution table (for example SYSTAB=ICONV) is used. The transliteration works recursively, that is if one of the substitution code points is not in the target set, this substitution will be used instead and so on.

```
  REPLACE GERMAN OE (0000D6=004F/0045)       WITH O E
  REPLACE GERMAN SZ (0000DF=0073/0073)       WITH s s
  REPLACE EURO MARK (0020AC=00D6/00DF/52/4F) WITH OE SZ R O
```

If you have a EURO sign in your text and convert it to *Latin1*, the resulting byte string will be *D6DF524F*. On the other hand, if you convert it to *ASCII*, the byte string will be *4F457373524F*.

The mapping as described above is always done and is also done recursive including the transliteration result. For example, if you define a mapping `(*30=39)` then the target data won't have any zero in the resulting text.

By replacing other code points recursively, you could easily cause infinite loops. To prevent this, the amount of replacements and the length of one replacement is limited to a maximum of 64. Note that this could still result in a large expansion of data. Therefore, be careful about defining recursive substitutions.

A sample user table for the ICONV system table to change the transliteration of German umlauts to AE, OE or UE is located in the SAMPLE directory under CCUTDEXL. Another sample user table called CCUTNPAS defines the *string.latin* subset (XOEF) which is mainly used for statutory reporting.

The order of definition is the order of processing the definitions. For exampel, if you define a mapping or a transliteration for code point X and later you make this code point invalid, then the code point is invalid. On the other hand, you can first deactivate all code points, then activate your subset and define your transliterations (best fit) or mappings.

For example: To define a non-expansion best fit mapping a user table can be used to delete all combined characters for single byte code pages. For the ICONV system transliteration table, a sample user table can be found under the name CCUTBFM1.

Some other subset user table definitions were added. For example:

- currently valid SEPA characters (<128) for money transfers (CCUTSEPA)
- characters valid in ISO-8859-15, CP1252 and IBM1142 (CCUTDELA)
- characters valid in ISO-8859-15, CP1252, IBM1142 and XOEF (CCUTDLAX)

All user table definitions are pre-calculated once at the beginning of execution to reduce the required CPU time for the conversion. So, the use of a user table increases the cost to open a file but has no effect on CPU utilization when converting the data.

To minimize the effort and to simplify the usage of user table, the most common subsets can be selected by keyword. In this case, a pre-calculated load module (DLL) is used. The subset definition can be for UTF (21 Bit UNICODE) or UCS (16 Bit UNICODE). If it is a UCS subset, then no valid code point is greater than 0xFFFF and you must use UCS CCSIDS to ensure that no code point greater than 0xFFFF is accepted.

If you have a working user table definition for a popular subset or system table manipulation, you can request a keyword for it. Please report this requirement over our issue tracking system

```
http://www.flam.de/en/technology/support/issues/
```

and attach the corresponding user table definition.

SELECTIONS

- `NPAS –UCS subset for statutory reporting (New passport, XOEF, String.Latin)`
- `SEPA –UCS subset of valid SEPA character smaller then 128`
- `DELA –UCS subset for German Latin of IBM1141, ISO8859-15 and CP1252`
- `DLAX –UCS subset for German Latin of IBM1141, ISO8859-15, CP1252 and XOEF`

4.2.246 PARAMETER *REPORTFILE*

SYNOPSIS

```
HELP:   Name of the report file to log substitutions
PATH:   CONV.WRITE.RECORD
TYPE:   STRING
SYNTAX: REPORTFILE='str'/STDOUT/STDERR
```

DESCRIPTION With the report capability, you can log every substitution (stop / ignore / substitute / translit / skip / ...) to a log file. For the report file, all rules and replacements are valid, which are described under "FILENAME HANDLING" above. For example:

```
REPORT=STDERR                           ; is SYSPRINT on z/OS
REPORT='~.MYREPORT(REPORT1)'            ; PDS on z/OS
REPORT='<SYSUID>.MYREPRT2'              ; PS dataset on z/OS
REPORT='~/myreport3.txt'                ; Unix path name
REPORT='<HOME>\reports\myreport4.txt'   ; Windows path name
```

A substitution report has the syntax listed below:

```
report_file         -> sub_definition_list
sub_definition_list -> sub_definition sub_definition_list
                    |  EMPTY
sub_definition      -> SUB '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    -> XLT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    |  IGN '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  STP '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  IDT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  SKP '(' UNIT ':' OFFSET '+' POSITION ':'
```

```
                          code_point byte_string ')'
                    |  UKW '(' UNIT ':' OFFSET '+' POSITION ':'
                          code_point byte_string ')'
  code_point_list   -> CODEPOINT '/' code_point_list
                    |  EMPTY
  code_point        -> CODEPOINT
                    |  EMPTY
  byte_string       -> '(' BYTESTRING ')'
                    |  EMPTY
```

A comment behind brackets explains the reason for substitution:

```
'invalid character'
'incomplete character'
'incomplete character at the end of a record'
```

The notations STP (stop), IGN (ignore), SUB (substitute), XLT (transliteration), IDT (identical) or SKP (skip) before round brackets indicate the action taken. If the task can not be determined then UKW (unknown) is used. A ignore of an invalid character is realized as substitution with nothing. All substitutions with a length of 0 are indicated with IGN (ignore). The position of such an abnormal/irregular character is coded inside the braces:

- UNIT number of records or blocks (depends on the data)
- OFFSET byte offset up to this unit in the complete data stream
- POSITION number of bytes inside the unit where the activity was done

All counting starts with 0. The first block or record has unit 0.

The position is relative to the data stream which was presented to the character conversion module. After conversions and formatting the position values can be far off from the input file which was read originally. For example, when reading a text file with text formatting to records, then delimiting characters are not included in the byte offset, the units are records and the position indicates which character was the issue for this log entry.

The CODEPOINTs (UNICODE) are taken from the defined substitution table The BYTESTRINGs are taken from the input (read characters) and output data (written characters). BYTESTRINGs and CODEPOINTs are provided if they are known in the cases of such an irregularity. In some cases, these values can be missing.

For interpreting the report file, you must know that only the values inside round brackets are meaningful. Text before or behind round brackets are comments, but this may be subject to changes.

SELECTIONS

- `STDOUT -Write output to stdout`
- `STDERR -Write output to stderr`

4.2.247 OVERLAY *COMPRESS*

SYNOPSIS

```
HELP:   Compression procedure [NONE]
PATH:   CONV.WRITE.RECORD
TYPE:   OVERLAY
SYNTAX: COMPRESS/CMP.{FLAM4()}
```

DESCRIPTION With the overlay compress you can choose which kind of compression method is used for the data. Depending on the compression method different parameter can be provided to define the behavior in more detail.

If no compression method selected no compression of the data will be done.

See below the different possibilities of compression.

4.2.248 OBJECT *FLAM4*

SYNOPSIS

```
HELP:   Compress records with FLAM
PATH:   CONV.WRITE.RECORD.COMPRESS
TYPE:   OBJECT
SYNTAX: FLAM4(MODE=CX7/CX8/VR8/ADC/NDC,COMMENT='str')
```

DESCRIPTION The FLAM4 compression to write compressed data into FLAMFILEs. As main argument you can define the compression mode (ADC, VR8, CX8). Additionally, it is possible to set a comment in the file header.

If the record-oriented method (write.record()) is used, the records with all attributes are written to the FLAM archive. If a block-oriented method (write.binary/char/text/xml()) is used, the data blocks are wrapped into records and written to the FLAMFILE as binary chunks with an undefined record format (U).

ARGUMENTS

- `NUMBER:MODE=CX7/CX8/VR8/ADC/NDC -Compression mode [ADC]`
  - `CX7 -Compression encoding as printable characters`
  - `CX8 -Compression encoding as binary data`
  - `VR8 -Enhanced compression encoding as binary data`
  - `ADC -Advanced Data Compression`
  - `NDC -No data compression`
- `STRING:COMMENT='str' -Comment for the member in the FLAMFILE [internal use]`

4.2.249 OVERLAY *ENCRYPT*

SYNOPSIS

```
HELP:   Encryption procedure [NONE]
PATH:   CONV.WRITE.RECORD
TYPE:   OVERLAY
SYNTAX: ENCRYPT/ENC.{F4PWD()/F4KME()}
```

DESCRIPTION The overlay *encrypt* allows you to choose which kind of encryption method is used for the data. Depending on the encryption method, different parameters can be set.

If no encryption method is selected, then no data encryption will be done.

See below for the encryption possibilities.

4.2.250 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password encryption
PATH:   CONV.WRITE.RECORD.ENCRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(MODE=AES/FLAM,PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Activates FLAM4 password-based encryption to encrypt the data written to FLAMFILEs. You can choose the encryption mode. The default and recommended mode is AES. If you provide no password, a hardcoded default password is used.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

If the data is already compressed, then MODE=NDC (no data compression) is used. Otherwise, ADC compression is used. If you set the compression mode to CX7/CX8/VR8 and enable encryption, e.g.

```
   comp.flam(mode=CX7/CX8/VR8) encr.f4pwd()
```

the compression mode is ignored and ADC/NDC is used instead. Due to how those three old compression modes work, they are incompatible with FLAM encryption.

ARGUMENTS

- NUMBER:MODE=AES/FLAM -Encryption mode [AES]
  - AES -AES encryption
  - FLAM -FLAM encryption
- STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']
  - DEFAULT -FLAM4 default password

4.2.251 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   CONV.WRITE.RECORD.ENCRYPT
TYPE:   OBJECT
SYNTAX: F4KME(MODE=AES/FLAM,LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 encryption of data written to FLAMFILEs based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME parameter is provided (encr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using encr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

You may define the encryption mode. The default and recommended mode is AES.

If the data is already compressed, then MODE=NDC (no data compression) is used. Otherwise, ADC compression is used to create the FLAMFILE. If you set the compression mode to CX7/CX8/VR8 and enable encryption, e.g.

```
comp.flam(mode=CX7/CX8/VR8) encr.f4kme()
```

the compression mode is ignored and ADC/NDC is used instead. Due to how those three old compression modes work, they are incompatible with FLAM encryption.

ARGUMENTS

- `NUMBER:MODE=AES/FLAM -Encryption mode [AES]`
  - `AES -AES encryption`
  - `FLAM -FLAM encryption`
- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME ['']`

4.2.252 OVERLAY *ENCODE*

SYNOPSIS

```
HELP:   Encoding procedure [NONE]
PATH:   CONV.WRITE.RECORD
TYPE:   OVERLAY
SYNTAX: ENCODE/COD.{BASE64()/BASE32()/BASE16()/OPGP()}
```

DESCRIPTION The overlay *encode* allows to choose which kind of encoding method is used for the data. Depending on the encoding method, different parameters can be set.

If no encoding method is selected, then no data encoding will be done.

See below for the available encoding options.

4.2.253 OBJECT *BASE64*

SYNOPSIS

```
HELP:   Base64 encoding (RFC4648)
PATH:   CONV.WRITE.RECORD.ENCODE
TYPE:   OBJECT
SYNTAX: BASE64(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record

boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- `NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]`
  - `NONE -No character set defined`
  - `SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)`
  - `ASCII -ASCII (mainly used in the for open system)`
  - `UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)`
  - `UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)`
  - `EBCDIC -EBCDIC (mainly used on IBM mainframe)`
  - `UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)`
  - `UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)`
  - `UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)`
  - `UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)`
  - `UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)`
  - `UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)`
  - `UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)`
  - `UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)`
- `NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]`
  - `BLOCK -Block-oriented conversion (all data is one chunk)`
  - `RECORD -Record-oriented conversion (convert each record individually)`

- NUMBER:LINE=num –Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM –Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST –Adds no delimiter (HOST)
  - BIN –Adds no delimiter (HOST)
  - NL –Adds delimiter new line (USS)
  - USS –Adds delimiter for USS (new line)
  - LF –Adds delimiter line feed (UNIX)
  - UNIX –Adds delimiter for UNIX (line feed)
  - CR –Adds delimiter carriage return (MAC)
  - OLDMAC –Adds delimiter for old MACs (carriage return)
  - CRLF –Adds delimiter carriage return line feed (WIN)
  - WINDOWS –Adds delimiter for WINDOWS (carriage return line feed)
  - DLM –Adds the system specific delimiter (DEFAULT)
  - SYSTEM –Adds the system specific delimiter (DEFAULT)
- SWITCH:CRCCHK –Add a CRC checksum to the base encoded data [FALSE]

4.2.254 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.RECORD.ENCODE.BASE64
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
```

```
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPCFBVFRMSVNUIGNvbW1l
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPCFBVFRMSVNUIGNvbW1l
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.255 OBJECT *BASE32*

SYNOPSIS

```
HELP:   Base32 encoding (RFC4648)
PATH:   CONV.WRITE.RECORD.ENCODE
TYPE:   OBJECT
SYNTAX: BASE32(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)

  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.256 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.RECORD.ENCODE.BASE32
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.257 OBJECT *BASE16*

SYNOPSIS

```
HELP:   Base16 encoding (RFC4648)
PATH:   CONV.WRITE.RECORD.ENCODE
TYPE:   OBJECT
SYNTAX: BASE16(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/ ↩
    UCS4BE/UTF32BE/UCS4LE/UTF32LE,MODE=BLOCK/RECORD,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/ ↩
    CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK,ARMOR())
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:MODE=BLOCK/RECORD -Encoding mode [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
- SWITCH:CRCCHK -Add a CRC checksum to the base encoded data [FALSE]

4.2.258 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   CONV.WRITE.RECORD.ENCODE.BASE16
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
```

```
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.2.259 OBJECT *OPGP*

SYNOPSIS

```
HELP:   Base64 encoding with 76 character, CRC checksum and Armor clasp (RFC4880)
PATH:   CONV.WRITE.RECORD.ENCODE
TYPE:   OBJECT
SYNTAX: OPGP(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE ↩
    /UTF32BE/UCS4LE/UTF32LE,DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM)
```

DESCRIPTION The overlay *encode* can be used to convert the output data using one of a couple of base encoding schemes, resulting in printable text. Base encoding data results in expansion of the data. The ratio of expansion depends on the base used.

Currently, Base16 (hexadecimal), Base32 and Base64 encodings are supported. Each of them is an object with further configurations options.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding can be performed in one of two modes:

The block mode converts the data as one large block of data. Padding will only be added at the end of the output file. This mode is the default for all write modes except for *write.record()*. This implies that record-based data will not retain the old record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* parameter specifies after how many characters a line is wrapped. The line delimiter can be configured via the *delim* parameter. By default, the system-specific delimiter is used.

When writing record-oriented (write.record()), each output line results in one record. Note that if the *line* parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

In record mode (default for wrote.record()), each output record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. A file that has been encoded in record mode, must also be read in record mode subsequently to decode the data correctly. Otherwise, the result might be garbage.

The character set used (ASCII (UTF-8) or EBCDIC) can also be configured. By default, the system-specific default character set is used.

Optionally, there is support for ASCII/EBCDIC armor headers and trailers around the base encoded data. This feature is mainly required to encode and decode PGP files (see RCF4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified.

The special OpenPGP encoding creates an ASCII armor output conforming to RFC4880 including the checksum. If you don't want the optional CRC24 checksum or if you want to use a line length other than 76, the Base64 encoding object with ASCII armor enabled should be used.

Additionally, a CRC checksum can be appended to the encoded data using the mechanism defined for OpenPGP. For Base64, CRC24 is used (RFC4480). For all other encoding schemas, CRC variants are used that match the schema's block size. Therefore, Base16 use CRC32 and Base32 uses CRC40, known from the GSM network. The CRC calculation is optional and independent of the data, encoding schema and the ASCII armor support. When reading, the verification is only done, if the corresponding switch is enabled, because it make no sense to consume CPU cycles for the CRC calculation and at the end, there is no checksum found to compare the calculated value.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and line>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)

- CR –Adds delimiter carriage return (MAC)
- OLDMAC –Adds delimiter for old MACs (carriage return)
- CRLF –Adds delimiter carriage return line feed (WIN)
- WINDOWS –Adds delimiter for WINDOWS (carriage return line feed)
- DLM –Adds the system specific delimiter (DEFAULT)
- SYSTEM –Adds the system specific delimiter (DEFAULT)

4.2.260 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for write operation
PATH:   CONV.WRITE.RECORD
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.2.261 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for write operation
PATH:   CONV.WRITE.RECORD
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- WIN –Choose Windows as platform (CRLF)

- UNX -Choose Unix derivative as platform (LF)
- ZOS -Choose ZOS as platform (NL)
- USS -Choose USS of ZOS as platform (NL)
- VSE -Choose VSE as platform (NL)
- BS2 -Choose BS2000-OSD as platform (NL)
- MAC -Choose MAC as platform (CR)

4.2.262 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of output data
PATH:   CONV.WRITE.RECORD
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)

  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.2.263 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   CONV.WRITE.RECORD.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.2.264 OBJECT *FLAM4*

SYNOPSIS

```
HELP:   Write records to a FLAM4FILE
PATH:   CONV.WRITE
TYPE:   OBJECT
SYNTAX: FLAM4(FILE='str',REMAIN,APPEND,NOPATH,SPACE(),CMPMODE=CX7/CX8/VR8/ADC/NDC,ENCMODE= ↩
    OFF/AES/FLAMENC,SPLIT.{},ORGANIZATION=SEQ/EDS/KDS/RDS/LDS,RECFORMAT=VAR/FIX/UND/TXT/DLM/ ↩
    VB/FB/V2B-XBE/V4B-XBE/V4A-ICL/V4E-ICL/V2B-ILE/V2B-IBE/V4B-IBE,RECLENGTH=num,ORGLENGTH= ↩
    num,KEYPOSITION=num,KEYLENGTH=num,COMMENT='str',PASSWORD/CRYPTOKEY='bin'/DEFAULT, ↩
    KMLIBRARY='str',KMFUNCTION='str',KMPARAMETER='str',CCSID='str'/DEFAULT/ASCII/EBCDIC,CASE ↩
    =UPPER/LOWER/FOLD/SUPPER/SLOWER/USRTAB,BOM,CHRMODE=STOP/IGNORE/SUBSTITUTE,SUBCHAR[num/ ↩
    SYSTEM...],SYSTABLE=ICONV,USRTABLE='str'/NPAS/SEPA/DELA/DLAX,REPORTFILE='str'/STDOUT/ ↩
    STDERR,LANG='str',PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH())
```

DESCRIPTION Writing with FLAM4 formats a list of elements into records and writes these records to a FLAMFILE of version 4 or older. If such a record is marked as text, a character set conversion from UTF-8 to the provided CCSID is done. If data elements are large blocks, then the result of the conversion is wrapped into records of the provided length or the default fixed length of 32,752 bytes.

During character conversion, the following options are supported:

- Stop, ignore or substitution of invalid or incomplete characters
- A custom substitution character list
- Different system substitution / transliteration tables
- A user-defined substitution table to override the system tables
- Case management (upper, lower, folding and special casing)
- Transliteration between subsets (mapping) and normalization
- BOM management and automatic detection of character sets
- Reporting of ignored or substituted characters

For the FLAMFILE archive, the following options are supported:

- Definition of record format and length for the FLAMFILE
- Definition of compression and encryption mode
- Correct ASA and machine print control character handling
- Pass phrase based or FKME based encryption technology
- If no FKME encryption is used, a comment can be stored

The member name will be the name of the original file from the read operation. The block mode will always be set to blocked, which means no unblocked FLAMFILEs can be generated. All the original file attributes are set automatically to the right values in the file header. User IO and pre- and post- processing exits are not supported. If these features are really required, the extended variant (XCNV/XDCO) of this command might be used.

ARGUMENTS

- `STRING:FILE='str' -Name/URL of the FLAMFILE to write ["==stdout]`
- `SWITCH:REMAIN -Remain existing files (don't overwrite) [FALSE]`
- `SWITCH:APPEND -Append the member to this FLAMFILE (not useful with encryption) [FALSE]`

- SWITCH:NOPATH -Exclude directory path of the original file in the compressed data [FALSE]
- NUMBER:CMPMODE=CX7/CX8/VR8/ADC/NDC -Compression mode [ADC]
  - CX7 -Compression encoding as printable characters
  - CX8 -Compression encoding as binary data
  - VR8 -Enhanced compression encoding as binary data
  - ADC -Advanced Data Compression
  - NDC -No data compression
- NUMBER:ENCMODE=OFF/AES/FLAMENC -Encryption mode [NONE]
  - OFF -Disable encryption
  - AES -AES encryption
  - FLAMENC -FLAM encryption
- NUMBER:ORGANIZATION=SEQ/EDS/KDS/RDS/LDS -Organization of the FLAMFILE [SEQ]
  - SEQ -Sequential access [DEFAULT]
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)
  - LDS -Linear data set (VSAM-LDS)
- NUMBER:RECFORMAT=VAR/FIX/UND/TXT/DLM/VB/FB/V2B-XBE/V4B-XBE/V4A-ICL/V4E-ICL/V2B-ILE/V2B-IBE/V4B-IBE -Record format for the FLAMFILE [FIX]
  - VAR -Variable
  - FIX -Fix
  - UND -Undefined
  - TXT -Text/Stream
  - DLM -Stream with special delimiter
  - VB -Variable blocked
  - FB -Fix blocked
  - V2B-XBE -Variable 2 byte length field without the length in big endian
  - V4B-XBE -Variable 4 byte length field without the length in big endian
  - V4A-ICL -Variable 4 byte length field in ASCII inclusive the length
  - V4E-ICL -Variable 4 byte length field in EBCDIC inclusive the length
  - V2B-ILE -Variable 2 byte length field inclusive the length in little endian
  - V2B-IBE -Variable 2 byte length field inclusive the length in big endian
  - V4B-IBE -Variable 4 byte length field inclusive the length in big endian
- NUMBER:RECLENGTH=num -Record length for the FLAMFILE [512]
- NUMBER:ORGLENGTH=num -Record length to wrap block oriented original data [32752]
- NUMBER:KEYPOSITION=num -Key position (>=1) for key sequential data sets [1]
- NUMBER:KEYLENGTH=num -Key length (>=1) for key sequential data sets [8]
- STRING:COMMENT='str' -Comment for the member in the FLAMFILE [internal use]
- STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['']
  - DEFAULT -FLAM4 default password
- STRING:KMLIBRARY='str' -Library name for FKME ['']
- STRING:KMFUNCTION='str' -Function name of FKME ['']
- STRING:KMPARAMETER='str' -Parameter for the call of FKME ['']

4.2.265 OBJECT *SPACE*

SYNOPSIS

```
HELP:   Space parameter for FLAMFILE allocation (mainframes only)
PATH:   CONV.WRITE.FLAM4
TYPE:   OBJECT
SYNTAX: SPACE(PRIMARY=num,SECONDARY=num,VOLSER='str',VOLUNIT='str',DATACLASS='str', ↩
    MGMTCLASS='str',STORCLASS='str',DISPSTATUS=NEW/SHR/MOD/OLD,NORMALDISP=CATALOG/DELETE/ ↩
    KEEP/UNCATALOG,ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG)
```

DESCRIPTION The FLAM4 file space allocation object collects several parameters for platforms where the cylinders or tracks, volume, classes and other things must be predefined.

The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

All size values must be given in megabyte (MiB). The caculation in the platform dependent units (tracks, cylinders, ...) are done inernally. Note: 1 MiB = 1024 KiB = 1024*1024 Byte

ARGUMENTS

- NUMBER:PRIMARY=num -Primary size in megabytes
- NUMBER:SECONDARY=num -Secondary size in megabytes
- STRING:VOLSER='str' -Volume serial number
- STRING:VOLUNIT='str' -Volume unit
- STRING:DATACLASS='str' -Data class
- STRING:MGMTCLASS='str' -Management class
- STRING:STORCLASS='str' -Storage class
- NUMBER:DISPSTATUS=NEW/SHR/MOD/OLD -Disposition status
  - NEW -Disposition status NEW (brand new file)
  - SHR -Disposition status SHR (read exiting file)
  - MOD -Disposition status MOD (append/modify file)
  - OLD -Disposition status OLD (overwrite existing file)
- NUMBER:NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for normal end
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for abnormal end
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog

4.2.266 OVERLAY *SPLIT*

SYNOPSIS

```
HELP:   Split parameter for FLAM4FILES (z/OS only)
PATH:   CONV.WRITE.FLAM4
TYPE:   OVERLAY
SYNTAX: SPLIT.{SIZE=num/NUMBER=num}
```

DESCRIPTION During compression a FLAMFILE can be splitted serially or in parallel into several parts, subject to the settings of the parameters SPLITMODE, SPLITNUMBER, and SPLITSIZE. Only the filename (or DD-name) of the first fragment of a split FLAMFILE must be specified at decompression and no additional settings are required. FLAM detects automatically whether and, if so, how a FLAMFILE has been splitted and searches by itself for the remaining fragments. Splitting of FLAMFILEs is only available for binary compression modes (MODE=CX8,VR8,ADC,NDC). Binary informations have been added to every part of a splitted FLAMFILE.

Serial Splitting Serial splitting (SPLIT.SIZE=nnnn) means that when the file currently used to store compressed data reaches a specified size limit it is closed and subsequent processing stores the compressed data into a newly created file (fragment). The number of fragments of a splitted FLAMFILE created is not limited. It only depends on the amount of data generated by the compression process. At decompression, FLAM verifies the order, the presence and the affiliation of all fragments.

This feature provides an efficient support for file size limitations (e.g. with e-mail attachments or file transfers). It can also improve system performance by allowing transmission of fragments over a network to begin before termination of the entire compression process.

Parallel Splitting With parallel splitting (SPLIT.NUMBER=n) compressed data is stored into a specified number of fragments. The current version can handle up to 4 parallel fragments. The size of the fragments depends on the amount of data generated during compression. At decompression, FLAM verifies the order, the presence and the affiliation of all fragments. Decompression requires the accessibility of all fragments of a FLAMFILE. None of the data can be recovered when one parallel fragment is missing.

One benefit of parallel splitting can consist in improved utilization of transmission capacities. Also, locally distributing FLAMFILE fragments may avoid unauthorized decompression without using encryption.

ARGUMENTS

- `NUMBER:SIZE=num -Size for serial splitting (after 1-4096 in megabytes)`
- `NUMBER:NUMBER=num -Number for parallel splitting (2-4 files)`

4.2.267 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion to this CCSID [DEFAULT]
PATH:   CONV.WRITE.FLAM4
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC
```

DESCRIPTION The TO code defines the encoding string or CCSID for the output text stream. The default value of this CCSID for read operations is UTF-8. For write operations, the default value is derived from the environment variable LANG. If LANG is not set, the default ASCII or EBCDIC system-codepage is used, depending on the platform.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
   CCSID='37'    will be equal to CCSID='IBM037'
   CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit. Optionally, you can use one of the keywords below to use the system-specific default ASCII or EBCDIC CCSID.

SELECTIONS

- `DEFAULT -Use default CCSID (system-specific)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`

4.2.268 PARAMETER *CASE*

SYNOPSIS

```
HELP:   Define upper, lower or special case conversion [NONE]
PATH:   CONV.WRITE.FLAM4
TYPE:   NUMBER
SYNTAX: CASE=UPPER/LOWER/FOLD/SUPPER/SLOWER/USRTAB
```

DESCRIPTION With the case modifier, you can perform conversion to upper or lower case as well as apply case folding and some special case conversions.

Case folding converts all characters to lower case but is more compatible with language specific rules for doing so. For example, this may be useful for case-insensitive string comparison if the strings contain non-Latin characters (code point < 128).

This mapping is done for each code point in front of substitution or transliteration. The respective case is accomplished on the basis of a separate Unicode conversion table in memory. This conversion can be manipulated if user mapping is activated and mapping definitions (*) are defined through the provided user table.

The options UPPER and LOWER convert characters without expansion, i.e. returned strings will never grow in length. Note, however, that case folding and the special options (convert e.g. ß to SS) can result in an expansion of data.

All these conversions are pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `UPPER -Change character to upper case`
- `LOWER -Change character to lower case`
- `FOLD -Change character to folded case (for compare)`
- `SUPPER -Change character to special upper case`
- `SLOWER -Change character to special lower case`
- `USRTAB -Change character according to user table`

4.2.269 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Write byte order mark in front of each record [FALSE]
PATH:   CONV.WRITE.FLAM4
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the write byte order mark keyword is defined, then the corresponding BOM sign is written at the start of the output stream (block oriented) or at the beginning of each record. If this BOM keyword is not used, then the BOM sign will not be written (this is valid independent of the chosen encoding/CCSID). If a valid BOM sign is located in the middle of the input stream, it is removed from the data. If you define the BOM keyword and the target charset contains only single bytes or no valid BOM sign, then no byte order mark will be written to the output file.

If an invalid BOM sign is recognized, then it will be ignored and causes no entry in the report file. If the byte order changes in the middle of text and the BOM handling switch is not activated, the conversion will stop and an error message is thrown.

By default, the write BOM keyword is set to OFF.

4.2.270 PARAMETER *CHRMODE*

SYNOPSIS

```
HELP:   Define handling of non-convertible characters [STOP]
PATH:   CONV.WRITE.FLAM4
TYPE:   NUMBER
SYNTAX: CHRMODE=STOP/IGNORE/SUBSTITUTE
```

DESCRIPTION With the MODE selection you can define how an invalid or incomplete character is handled inside of a unit by this component. All possible modes are listed at the end of this chapter.

This MODE is valid for the content of each unit (block or record). At the end of a unit there can be an incomplete character. Such a rest will be handled dependent on the METHOD you choose. The rest of the last unit of a data field will be concatenated to the first unit of the next data field. For all other records in a data field the handling of such a rest will be defined by the respective METHOD.

The SKIP case for METHOD is only relevant for incomplete byte sequences at the end of a unit. The ignore and substitution mode is only effective for all characters in the middle of the unit, besides an incomplete byte sequence occurs in the last unit. Please don't mix up the IGNORE case for MODE with the SKIP case of METHOD.

If a system or user substitution table is used the stop, ignore or substitute are only relevant for characters not managed by the CCSIDs and the substitution tables. This means that the USRTAB and SYSTAB are added to the capabilities of character conversion and only for the rest the chosen MODE is used.

4.2.270.1 CONSTANT *STOP*

SYNOPSIS

```
HELP:   Stop at the first non-convertible character
PATH:   CONV.WRITE.FLAM4.CHRMODE
TYPE:   NUMBER
SYNTAX: STOP
```

DESCRIPTION Stops conversion at the first incomplete or invalid character which can not be converted. An error message is given with the position of abortion and the value in the input stream. An entry will be (STP()) written to the report file to log this event. To be compatible with other conversion tools, this is the default case, but we recommend to use IGNORE or SUBSTITUTE to prevent any abortion.

indexterm:[Constant STOP}

4.2.270.2 CONSTANT *IGNORE*

SYNOPSIS

```
HELP:   Ignore non-convertible characters
PATH:   CONV.WRITE.FLAM4.CHRMODE
TYPE:   NUMBER
SYNTAX: IGNORE
```

DESCRIPTION Ignores incomplete or invalid characters and writes a report entry for each character. IGNORE is a substitution by no corresponding character that means no substitution table is required.

indexterm:[Constant IGNORE}

4.2.270.3 CONSTANT *SUBSTITUTE*

SYNOPSIS

```
HELP:   Substitute non-convertible characters
PATH:   CONV.WRITE.FLAM4.CHRMODE
TYPE:   NUMBER
SYNTAX: SUBSTITUTE
```

DESCRIPTION Substitutes incomplete or invalid character and writes a report entry for each character. For SUBSTITUTE you can define a substitution character list (SUBCHR). If no SUBCHR is provided then this option substitutes invalid or incomplete characters with code point 0x00001A. If a SUBCHR is defined all invalid or incomplete characters will be substituted by the defined list of code points that can be up to 8. If you specify a SUBCHR the substitution mode will be enabled.

indexterm:[Constant SUBSTITUTE}

4.2.271 PARAMETER *SUBCHAR*

SYNOPSIS

```
HELP:   Default substitution character list [0x1A] (maximal 8 code points)
PATH:   CONV.WRITE.FLAM4
TYPE:   NUMBER
SYNTAX: SUBCHAR[num/SYSTEM...]
```

DESCRIPTION As substitution character (SUBCHR) you can define a list with up to 8 UNICODE points which are used for the substitution of invalid or incomplete characters if the corresponding code point is not defined in the substitution table (see SUBTAB). If you assign to it the keyword SYSTEM the value 0x1A is written as default substitution character.

If you activate substitution and no SUBCHR is defined, then unsupported characters will be replaced by CODEPOINT 0x00001A.

On the other hand SUBCHR will replace every undefined code point in the substitution table, what is the same behavior when using a USRTAB with corresponding SUBCHR values for every possible code point (000000-200000). If SUBCHR but no MODE is defined the substitution mode will be activated.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `SYSTEM -Use system specific substitution char (0x1A for ASCII/UTF, 0x3F for EBCDIC)`

4.2.272 PARAMETER *SYSTABLE*

SYNOPSIS

```
HELP:   Define a preloaded system substitution table
PATH:   CONV.WRITE.FLAM4
TYPE:   NUMBER
SYNTAX: SYSTABLE=ICONV
```

DESCRIPTION With the system table (SYSTAB) you can load a substitution table with a predefined transliteration schema. Entries in the USRTAB can be used to override predefined transliteration entries or add missing transliteration code points. The options listed at the end of this chapter are available.

If a substitution table is used then the MODE option (STOP/IGNORE/SUBSTITUTE) is only relevant for characters that cannot be converted or substituted.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `ICONV -Preload with ICONV//TRANSLIT substitution table`

4.2.273 PARAMETER *USRTABLE*

SYNOPSIS

```
HELP:   Name of the file containing the user conversion table
PATH:   CONV.WRITE.FLAM4
TYPE:   STRING
SYNTAX: USRTABLE='str'/NPAS/SEPA/DELA/DLAX
```

DESCRIPTION With this parameter, you can choose a predefined subset or standard manipulation for the system tables via keyword (see `flcl help "xcnv.inp .sav.fil.cnv.chr.usrtab"` for available keywords). Alternatively, you can use a file containing a user defined substitution table (USRTAB).

For the user table file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   USRTAB=NPAS                          ; use XOEV predefined subset
   USRTAB=DELA                          ; German Latin single byte subset
   USRTAB='~.MYUSRTAB(USRTAB1)'         ; PDS on z/OS
   USRTAB='<SYSUID>.MYUSRT02'           ; PS dataset on z/OS
   USRTAB='~/myusrtab3.txt'             ; Unix path name
   USRTAB='<HOME>\usrtab\myusrtab4.txt' ; Windows path name
```

A user defined substitution table has the syntax listed below:

```
   usr_tab_file         -> usr_definition_list
   usr_definition_list  -> usr_definition usr_definition_list
                        |  EMPTY
   usr_definition       -> '(' codepoint_definition ')'
   codepoint_definition -> cp_activation
                        |  cp_deactivation
                        |  cp_transliteration
                        |  cp_standard_mapping
                        |  cp_usr_case_mapping
   cp_activation        -> ['+'] CODEPOINT  [ '-' CODEPOINT ]
   cp_deactivation      ->  '-'  CODEPOINT  [ '-' CODEPOINT ]
   cp_transliteration   -> ['+'] CODEPOINT '=' code_point_list
   cp_standard_mapping  ->  '*'  CODEPOINT '=' code_point_list
   cp_usr_case_mapping  ->  '^'  CODEPOINT '=' code_point_list
   code_point_list      -> CODEPOINT separator code_point_list
                        |  CODEPOINT
                        |  EMPTY
   separator            -> '/'
                        |  ','
                        |  ';'
```

Note: In the description below "/" is used as code point separator.

The optional signs in front of a code point (CP) have the following meaning:

```
   '+' - a valid transliteration (only if CP not in the target set)
   '*' - a valid mapping         (this translation is always done)
   '^' - a case mapping/folding  (only if case=usrtab is activated)
   '-' - an invalid definition   (removes CP for subset definitions)
```

If no sign is used, then a transliteration for a valid code point is defined. Valid code points are accepted, invalid code points result in in an error. Error handling depends on the mode (STOP,IGNORE, SUBSTITUTE). Definition of + code points without code point list, thus without = is equal definition of valid code points. With an asterisk *, you can define an enforced mapping for this code point. For example, this can be used to delete this character if no code point list is provided or to translate this character always in another value. This can also be used to convert code points outside of a subset into this subset.

In contrast to the transliteration (+ is only done if a character doesn't exist in the target encoding), the mapping is always done for each character. Invalid (−) code points can be used to deactivate a code point. To activate (+) or deactivate (−) a range of code points you can use the minus mark (−) followed by one sub code point. The range goes from the smaller code point to the bigger one. For example:

```
(-00-7F) deactivate all US-ASCII code points
(+39-30) activate all decimal digits
```

The range operator (−) with the optional plus sign (+) can also be used to activate code points.

To define an own case mapping or folding (required for certain subsets) the caret sign (^) can be used. This mapping is only done if the user table is activated for case mapping (CASE=USRTAB).

To define transliterations/substitutions/mappings an assignment (=) of a code point list is required. The code point list can contain a maximum of 8 code points. If no code point list is specified, then the character is ignored. A transliteration to itself simply activates this code point. The same is valid for mapping definitions. A code point is a hexadecimal number representing a 21 bit UNICODE point.

```
00000000 to 001FFFFF hex
```

If a code point is not defined in the substitution table (USRTAB and/or SYSTAB), the appropriate substitution character (SUBCHR) is used. If SUBCHR is not set, no substitution is performed. Depending on the MODE, character conversion stops with an error (STOP) or ignores the character (IGNORE).

Text before and after brackets are comments.

Between "(" and "=" or "/" or ")" you can define hex digits until the first non-hex digit. All non-hex digits up to the next separator are interpreted as comment. Leading whitespace is ignored.

```
  REPLACE GERMAN SZ (00DF=0073/0073)                  WITH ss
THIS IS AN EXAMPLE FOR COMMENTS IN CODE POINT LIST:
  REPLACE EURO MARK (20AC= 45#E / 55#U / 52#R / 4F#O) WITH EURO
  REPLACE BOM MARK  (EFFF=)                           WITH NOTHING
```

Leading zeros are allowed. If you don't supply a hex value, then 0x00 is used.

```
  REPLACE GERMAN SZ (00DF=/)                   WITH 0x00 0x00
  REPLACE EURO MARK (20AC=00000045/00055/000052/000004F) WITH EURO
```

ATTENTION: Please don't use parentheses "()" or the operators in your comments.

To describe your own subsets, you can use a user table without a system table. The USRTAB can also be used to overwrite or add transliterations when a system substitution table (for example SYSTAB=ICONV) is used. The transliteration works recursively, that is if one of the substitution code points is not in the target set, this substitution will be used instead and so on.

```
  REPLACE GERMAN OE (0000D6=004F/0045)       WITH O E
  REPLACE GERMAN SZ (0000DF=0073/0073)       WITH s s
  REPLACE EURO MARK (0020AC=00D6/00DF/52/4F) WITH OE SZ R O
```

If you have a EURO sign in your text and convert it to *Latin1*, the resulting byte string will be *D6DF524F*. On the other hand, if you convert it to *ASCII*, the byte string will be *4F457373524F*.

The mapping as described above is always done and is also done recursive including the transliteration result. For example, if you define a mapping `(*30=39)` then the target data won't have any zero in the resulting text.

By replacing other code points recursively, you could easily cause infinite loops. To prevent this, the amount of replacements and the length of one replacement is limited to a maximum of 64. Note that this could still result in a large expansion of data. Therefore, be careful about defining recursive substitutions.

A sample user table for the ICONV system table to change the transliteration of German umlauts to AE, OE or UE is located in the SAMPLE directory under CCUTDEXL. Another sample user table called CCUTNPAS defines the *string.latin* subset (XOEF) which is mainly used for statutory reporting.

The order of definition is the order of processing the definitions. For exampel, if you define a mapping or a transliteration for code point X and later you make this code point invalid, then the code point is invalid. On the other hand, you can first deactivate all code points, then activate your subset and define your transliterations (best fit) or mappings.

For example: To define a non-expansion best fit mapping a user table can be used to delete all combined characters for single byte code pages. For the ICONV system transliteration table, a sample user table can be found under the name CCUTBFM1.

Some other subset user table definitions were added. For example:

- currently valid SEPA characters (<128) for money transfers (CCUTSEPA)
- characters valid in ISO-8859-15, CP1252 and IBM1142 (CCUTDELA)
- characters valid in ISO-8859-15, CP1252, IBM1142 and XOEF (CCUTDLAX)

All user table definitions are pre-calculated once at the beginning of execution to reduce the required CPU time for the conversion. So, the use of a user table increases the cost to open a file but has no effect on CPU utilization when converting the data.

To minimize the effort and to simplify the usage of user table, the most common subsets can be selected by keyword. In this case, a pre-calculated load module (DLL) is used. The subset definition can be for UTF (21 Bit UNICODE) or UCS (16 Bit UNICODE). If it is a UCS subset, then no valid code point is greater than 0xFFFF and you must use UCS CCSIDS to ensure that no code point greater than 0xFFFF is accepted.

If you have a working user table definition for a popular subset or system table manipulation, you can request a keyword for it. Please report this requirement over our issue tracking system

```
http://www.flam.de/en/technology/support/issues/
```

and attach the corresponding user table definition.

SELECTIONS

- `NPAS –UCS subset for statutory reporting (New passport, XOEF, String.Latin)`
- `SEPA –UCS subset of valid SEPA character smaller then 128`
- `DELA –UCS subset for German Latin of IBM1141, ISO8859-15 and CP1252`
- `DLAX –UCS subset for German Latin of IBM1141, ISO8859-15, CP1252 and XOEF`

4.2.274 PARAMETER *REPORTFILE*

SYNOPSIS

```
HELP:   Name of the report file to log substitutions
PATH:   CONV.WRITE.FLAM4
TYPE:   STRING
SYNTAX: REPORTFILE='str'/STDOUT/STDERR
```

DESCRIPTION With the report capability, you can log every substitution (stop / ignore / substitute / translit / skip / . . .) to a log file. For the report file, all rules and replacements are valid, which are described under "FILENAME HANDLING" above. For example:

```
REPORT=STDERR                          ; is SYSPRINT on z/OS
REPORT='~.MYREPORT(REPORT1)'           ; PDS on z/OS
REPORT='<SYSUID>.MYREPRT2'             ; PS dataset on z/OS
REPORT='~/myreport3.txt'               ; Unix path name
REPORT='<HOME>\reports\myreport4.txt'  ; Windows path name
```

A substitution report has the syntax listed below:

```
report_file         -> sub_definition_list
sub_definition_list -> sub_definition sub_definition_list
                    |  EMPTY
sub_definition      -> SUB '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    -> XLT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    |  IGN '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  STP '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  IDT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  SKP '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  UKW '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
code_point_list     -> CODEPOINT '/' code_point_list
                    |  EMPTY
code_point          -> CODEPOINT
                    |  EMPTY
byte_string         -> '(' BYTESTRING ')'
                    |  EMPTY
```

A comment behind brackets explains the reason for substitution:

```
'invalid character'
'incomplete character'
'incomplete character at the end of a record'
```

The notations STP (stop), IGN (ignore), SUB (substitute), XLT (transliteration), IDT (identical) or SKP (skip) before round brackets indicate the action taken. If the task can not be determined then UKW (unknown) is used. A ignore of an invalid character is realized as substitution with nothing. All substitutions with a length of 0 are indicated with IGN (ignore). The position of such an abnormal/irregular character is coded inside the braces:

- UNIT number of records or blocks (depends on the data)
- OFFSET byte offset up to this unit in the complete data stream
- POSITION number of bytes inside the unit where the activity was done

All counting starts with 0. The first block or record has unit 0.

The position is relative to the data stream which was presented to the character conversion module. After conversions and formatting the position values can be far off from the input file which was read originally. For example, when reading a text file with text formatting to records, then delimiting characters are not included in the byte offset, the units are records and the position indicates which character was the issue for this log entry.

The CODEPOINTs (UNICODE) are taken from the defined substitution table The BYTESTRINGs are taken from the input (read characters) and output data (written characters). BYTESTRINGs and CODEPOINTs are provided if they are known in the cases of such an irregularity. In some cases, these values can be missing.

For interpreting the report file, you must know that only the values inside round brackets are meaningful. Text before or behind round brackets are comments, but this may be subject to changes.

SELECTIONS

- `STDOUT -Write output to stdout`
- `STDERR -Write output to stderr`

4.2.275 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for write operation
PATH:   CONV.WRITE.FLAM4
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.2.276 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for write operation
PATH:   CONV.WRITE.FLAM4
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.2.277 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of output data
PATH:   CONV.WRITE.FLAM4
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.2.278 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   CONV.WRITE.FLAM4.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.2.279 OBJECT *DIR*

SYNOPSIS

```
HELP:   Defines how to handle directories and various file types if wildcards used
PATH:   CONV
TYPE:   OBJECT
SYNTAX: DIR(LINK,ALIAS,HIDDEN,RECURSIVE,ARCHIVE,ONEFILESYSTEM,TAPE)
```

DESCRIPTION This object defines multiple switches that control how input files are found. This is especially important when using wildcards in input filenames.

There are switches to enable recursion into subdirectories, following symbolic links, finding hidden files, recursion into archive files, include files on tapes, resolving aliases and enforcing to not leave the filesystem.

If the recursive flag is set, the filename portion of the input file string is searched for recursively. If a directory path containing wildcards is present, the filename portion is searched for recursively in all paths that match the directory path pattern, e.g. "**/path/**.txt" matches my/path/hugo.txt, another/path/hugo.txt as well as my/path/subdir/hugo.txt. Without the recursive flag, the last file would not be found.

If the archive switch is enabled, FLAMFILES, TARBALLS, ZIP files and other kinds of supported archives are searched. All the matching members are extracted. In other words, archives are interpreted like subdirectories if the flag is enabled.

If the tape switch is enabled, then datasets stored on tapes are searched. If the name of such a file matches the pattern, then it might occur that console input and a manual intervention is required to retrieve the respective dataset(s).

If the link switch is enabled, symbolic links are followed (otherwise they are ignored). Links are mainly supported on modern file systems. For dataset aliases on mainframe systems, see the alias flag.

With the alias switch enabled, datasets names which are aliases of the real dataset are resolved to their actual names (otherwise they are ignored). Dataset aliases only exist on mainframe systems. This flag has no effect on other platforms.

If a transparent read method (e.g. read.auto()) is used, then the archive flag is enabled by default and cannot be disabled.

ARGUMENTS

- `SWITCH:LINK -Include symbolic links`
- `SWITCH:ALIAS -Resolve data set aliases`
- `SWITCH:HIDDEN -Include hidden files`
- `SWITCH:RECURSIVE -Include sub directories`
- `SWITCH:ARCHIVE -Include archives (FLAMFILEs, ZIP, ...)`
- `SWITCH:ONEFILESYSTEM -Don't cross file system boundaries`
- `SWITCH:TAPE -Include files deposited on tapes`

4.2.280 OVERLAY *LOGGING*

SYNOPSIS

```
HELP:   Select target for logging
PATH:   CONV
TYPE:   OVERLAY
SYNTAX: LOGGING.{STREAM()/FILE()/SYSTEM()}
```

DESCRIPTION The log destination parameter defines where the log messages will be written.

Each log message starts with the current time and date.

All the possible destinations allow to specify a custom identification string which will be printed after time and date of each message. This is done with the IDENT argument.

If no IDENT specified the default will be *owner.program.command*

If no log destination defined stream will be used as default.

4.2.281 OBJECT *STREAM*

SYNOPSIS

```
HELP:   Log to a stream
PATH:   CONV.LOGGING
TYPE:   OBJECT
SYNTAX: STREAM(FORMAT=STANDARD/DIALOG,IDENT='str',STDOUT)
```

DESCRIPTION The STREAM object allows the log messages to be written to the standard output or the standard error output. Standard error output is used if STDOUT is not specified.

ARGUMENTS

- `NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]`
  - `STANDARD -Standard logging format (timestamp ident *level* message)`
  - `DIALOG -For dialog processing (ident *level* message)`
- `STRING:IDENT='str' -Identifier used to identify log entries`
- `SWITCH:STDOUT -Log messages to STDOUT instead of STDERR`

4.2.282 OBJECT *FILE*

SYNOPSIS

```
HELP:   Log to a flat file
PATH:   CONV.LOGGING
TYPE:   OBJECT
SYNTAX: FILE(FORMAT=STANDARD/DIALOG,IDENT='str',NAME='str')
```

DESCRIPTION The flat file option allows the log messages to be written to a normal file. The name of the file must be specified for this destination. The log messages will be appended to this file if it already exists.

For the log file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   NAME='~.MYLOG(LOG1)'                    ; PDS on z/OS
   NAME='<SYSUID>.MYLOG02'                 ; PS dataset on z/OS
   NAME='~/mylog3.txt'                     ; Unix path name
   NAME='<HOME>\logs\mylog4.txt'           ; Windows path name
```

ATTENTION: Parallel process cannot share the same log file

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- STRING:NAME='str' -Filename for appending log messages

4.2.283 OBJECT *SYSTEM*

SYNOPSIS

```
HELP:   Use system logging facility
PATH:   CONV.LOGGING
TYPE:   OBJECT
SYNTAX: SYSTEM(IDENT='str')
```

DESCRIPTION The SYSTEM destination allows the log messages to be written to the logging system of the operating system.

On Unix and z/OS® systems the syslog() call is used. On Windows® the event logging facility is used.

```
NOTE: On Windows(R) this requires the registry key entry:
      HKEY_LOCAL_MACHINE
        SYSTEM
          CurrentControlSet
            Services
              EventLog
                Application
                  FLM
```

This registry key will be created by using the DLL flevent.dll

ARGUMENTS

- STRING:IDENT='str' -Identifier used to identify log entries

4.2.284 OBJECT *MESSAGE*

SYNOPSIS

```
HELP:   Select type of messages to log
PATH:   CONV
TYPE:   OBJECT
SYNTAX: MESSAGE(NONE,ERROR,ERRTRACE,WARNING,NOTICE,DEBUG,ABOUT,VERSION,INPUT,PARAMETER, ↩
    STATISTIC,INFO,LICID,LICENSE,PROGRESS,ALL,DEFAULT,MINIMAL,ERRONLY)
```

DESCRIPTION The log message specification is a classification of the messages to write to the selected log destination. Each type is selectable individually by adding its argument inside this object.

For added convenience, the special types NON, ALL, MINIMAL select the proper message types with just one argument.

If no message specification is given, a DEFAULT is used which writes out messages of these types:

- ERROR
- ERRTRACE
- WARNING
- NOTICE
- PARAMETER
- STATISTIC
- INFO
- LICID

ARGUMENTS

- `SWITCH:NONE -No messages will be logged`
- `SWITCH:ERROR -Error messages will be logged`
- `SWITCH:ERRTRACE -Error trace will be logged`
- `SWITCH:WARNING -Warnings will be logged`
- `SWITCH:NOTICE -Notices will be logged`
- `SWITCH:DEBUG -Debug information will be logged`
- `SWITCH:ABOUT -About information will be logged`
- `SWITCH:VERSION -Version information will be logged`
- `SWITCH:INPUT -Input data will be logged (dangerous for passwords)`
- `SWITCH:PARAMETER -Parsed parameters will be logged`
- `SWITCH:STATISTIC -Statistics will be logged`
- `SWITCH:INFO -Requested information will be logged`
- `SWITCH:LICID -License ID will be logged`
- `SWITCH:LICENSE -License summary will be logged`
- `SWITCH:PROGRESS -Progress bar will be logged`
- `SWITCH:ALL -All messages are logged`
- `SWITCH:DEFAULT -Relevant messages are logged`
- `SWITCH:MINIMAL -Only important messages are logged`
- `SWITCH:ERRONLY -Only errors and error trace are logged`

4.2.285 PARAMETER *INVERSE*

SYNOPSIS

```
HELP:   Generate a XCNV command file (xcnv "out=thisfile") for inverse conversion [NONE]
PATH:   CONV
TYPE:   STRING
SYNTAX: INVERSE='str'/STDOUT/STDERR
```

DESCRIPTION With the INVERSE parameter you can generate an XCNV command file that inverts the operations executed by the specified CONV command.

For the inverse file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   INVERSE=STDERR                          ; is SYSPRINT on z/OS
   INVERSE='~.MYINVERS(INVERS1)'           ; PDS on z/OS
   INVERSE='<SYSUID>.MYIVRS02'             ; PS dataset on z/OS
   INVERSE='~/myinvers3.txt'               ; Unix path name
   INVERSE='<HOME>\usrtab\myinvers4.txt'   ; Windows path name
```

This can be used, for example, if you need a solution to edit a file inside an archive and the file to be edited is in a format that is not supported by the system (e.g. an ASCII text file on an EBCDIC system). With the INVERSE parameter of the CONV command, you can extract the file from the archive, do necessary conversions and write it to a temporary file. After editing, you pass the generated XCNV command file to the XCNV command which reverses all operations performed by CONV, i.e. the text file is converted back to its original form and written back to the archive.

Below you can see an example of how it works:

```
   * <DSN> - Dataset name to edit
   * <TMP> - Temporary file for editing
   * <IVR> - Command file to inverse write back

   flcl conv read.file=<DSN> write.record(file=<TMP>) inverse=<IVR>
   edit(<TMP>)
   flcl xcnv=<IVR>
```

There are many other possibilities to use the INVERSE parameter. This sample is implemented in the FLEDIT line command for ISPF. This allows you to correct TEXT and XML files with the normal ISPF editor even though they may be encoded, compressed or in another codepage not supported by the editor.

The inverse command reverses what the call of CONV produces. This could be different from the original. For example, if you use

```
read.auto() write.record()
```

on an XML file, then pretty printing is used to form the records. The inverse command converts back the pretty printed XML. The result would be the same XML document but with a different text formatting. If the file is a text file, write.record() replaces the delimiter with record length information. Since the original delimiter was lost, the inverse command adds the system delimiter. If the original file was delimited with 0x0D0A and you use the inversion feature on a non-Windows system, the original delimiter will be replaced by the correct delimiter for the current system and is then translated to the original character set. The same will happen if you read a base encoded file. This file can be in ASCII or EBCDIC at read, but it will be written in the system specific character set. This behavior ensures, that the written file is a correct base encoded file for this platform.

SELECTIONS

- `STDOUT -Write output to stdout`
- `STDERR -Write output to stderr`

4.3 COMMAND *XCNV*

SYNOPSIS

```
HELP:   Extended data conversion
PATH:   LIM.flcl
TYPE:   OBJECT
SYNTAX: :> flcl XCNV(INPUT(),OUTPUT(),DIR(),LOG())
```

DESCRIPTION The XCNV command reads and writes different data formats on different platforms. It use the read and write routines for original data of FLAM. More precisely, it provides all conversion capabilities of FLAM without a FLAM archive in the background.

The XCNV command was designed to provide the full capacity of the powerful Frankenstein Limes Universal Converter (FLUC). It can be used to read locally or remote data sources of different kind and write locally or remote this data in the same or different formats. The XCNV command can for example be used to read an encoded, encrypted, compressed text file in UTF-16 from an open platform and write the text records in a mainframe data set with a dedicated record format (FB, VB, ...) in EBCDIC with ASA or machine print control character or wise versa. At this you must know which kind of data you read, you can manipulate each kind of attribute and you have the complete control over each conversion step.

During this it supports:

- Different kind of remote access protocols (IP-Nativ, MQ-Series)
- Different kind of data sources (Files, Streams, Tables, Archives)
- Different kinds of IO methods (Binary, Character, Text, Record, FLAM)
- Different encodings of the data (HEX (BASE16), BASE32, BASE64)
- Different encryption methods (FLAM, PGP)
- Different compression methods (GZIP, BZIP2, XZ)
- Powerful character conversion based on CCSIDs
- Different logical data formatting (BIN, TEXT, XML)

If the XCNV command is to complex or you need auto detection capabilities you can use the simplified variant called CONV which also provides the full conversion capabilities of FLAM but with more automatism. The CONV command provides a easy to handle subset of the total possible conversions. In contrast to XCNV the CONV command does a lot of auto detection and automated conversions. It try to presage how the data must be represent on the current system. For XCNV command the user must know in detail what kind of data is read and can full control/ define how the data is written.

To get syntax information, please use:

```
flcl SYNTAX XCNV
```

To get help for a parameter, please use:

```
flcl HELP XCNV.parameter[.parameter[...]]
```

To read the manual page for a parameter, please use:

```
flcl MANPAGE XCNV.parameter[.parameter[...]]
   or
flcl HELP XCNV.parameter[.parameter[...]] MAN
```

To generate the user manual for the command, please use:

```
flcl GENDOCU XCNV=filename
```

Parameters can be defined by the command line (directly or per file) or by properties taken from the corresponding property file

EXAMPLE

```
   flcl XCNV inp(sav.fil(fio.blk(file='my.file')
                         cnv.hsh(format=bsd)
                         cnv.gzp()
                         cnv.chr(from='UTF-8' to='IBM1141')
                         fmt.txt(suptws)))
             out(sav.fil(fmt.txt(method=HST)
                         cnv.hsh(algo=SHA512)
                         fio.rec(file='my.output'
                                 falloc(orga=SEQ,recf=VB,recl=512))
```

Reads file as binary blocks, generates a SHA1 checksum in BSD style, performs GZIP decompression, converts characters from UTF-8 to IBM1141 and formats the data into text records and rest elements in the read operation. On write, the text elements are formatted without a delimiter, a SHA-512 checksum is calculated for the result and the data elements are eventually written to a record-oriented PS-VB-512 dataset.

4.3.1 OBJECT *INPUT*

SYNOPSIS

```
HELP:   Inbound parameter
PATH:   XCNV
TYPE:   OBJECT
SYNTAX: INPUT(NET.{},SAV.{})
```

DESCRIPTION Over the object INPUT you can define how the data is read.

4.3.2 OVERLAY *NET*

SYNOPSIS

```
HELP:   Network [NONE]
PATH:   XCNV.INPUT
TYPE:   OVERLAY
SYNTAX: NET.{IPN()/IPS()/MQS()}
```

DESCRIPTION With the overlay NET you can choose the protocol for the remote read or write access. Below You can find the currently supported protocols and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.

4.3.3 OBJECT *IPN*

SYNOPSIS

```
HELP:   IP-Native
PATH:   XCNV.INPUT.NET
TYPE:   OBJECT
SYNTAX: IPN(HOST='str',PORT='str',USER='str',AUTH='str')
```

DESCRIPTION The IP native (IPN) method supports IPv4 and IPv6 network communication for remote access to files, archives and other sources and targets of the FLAM infrastructure. For this client-server architecture, the FLIES server must be started and running in the IPN mode listening on a dedicated port (17996) on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or DNS name) and the corresponding service port (number or name) must be specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES server is required to handle requests.

ARGUMENTS

- `STRING:HOST='str' -IP address [localhost]`
- `STRING:PORT='str' -Port number [17996]`
- `STRING:USER='str' -User name [optional]`
- `STRING:AUTH='str' -Authentication data [optional]`

4.3.4 OBJECT *IPS*

SYNOPSIS

```
HELP:   IP-Secure (SSL/TLS)
PATH:   XCNV.INPUT.NET
TYPE:   OBJECT
SYNTAX: IPS(HOST='str',PORT='str',USER='str',AUTH='str')
```

DESCRIPTION The IP secure (IPS) method supports IPv4 and IPv6 network communication with SSL/TLS for remote access to files, archives and other sources and targets of the FLAM infrastructure. For this client-server architecture, the FLIES server must be started and running in the IPS mode listening on a dedicated port (17996) on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or DNS name) and the corresponding service port (number or name) must be specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES server is required to handle requests.

ARGUMENTS

- `STRING:HOST='str' -IP address [localhost]`
- `STRING:PORT='str' -Port number [17996]`
- `STRING:USER='str' -User name [optional]`
- `STRING:AUTH='str' -Authentication data [optional]`

4.3.5 OBJECT *MQS*

SYNOPSIS

```
HELP:   MQ-Series
PATH:   XCNV.INPUT.NET
TYPE:   OBJECT
SYNTAX: MQS(QMANAGER='str',QUEUE='str',USER='str',AUTH='str')
```

DESCRIPTION The MQ-Series (MQS) method supports IBM® message queuing for remote access to files, archives and other sources and targets of the FLAM infrastructure. For this client-server architecture, the FLIES server must be started and running in MQS mode listening on a dedicated queue on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES server is required to handle requests.

ARGUMENTS

- `STRING:QMANAGER='str' -Queue manager name [""]`
- `STRING:QUEUE='str' -Queue name [""]`
- `STRING:USER='str' -User name [optional]`
- `STRING:AUTH='str' -Authentication data [optional]`

4.3.6 OVERLAY *SAV*

SYNOPSIS

```
HELP:   Storage [FILE]
PATH:   XCNV.INPUT
TYPE:   OVERLAY
SYNTAX: SAV.{FILE()}
```

DESCRIPTION With the overlay SAV you can choose the kind of source or target for the read or write operation. Below You can find the currently supported procedures and the corresponding parameter explained in a dedicated chapter.

4.3.7 OBJECT *FILE*

SYNOPSIS

```
HELP:   File handling
PATH:   XCNV.INPUT.SAV
TYPE:   OBJECT
SYNTAX: FILE(FIO.{},CNV[{}...],FMT.{},ENV[()...])
```

DESCRIPTION Over the object FILE you can define how a list of neutral FLAM5 elements with type, length, value and attributes is read from a file. For this you can choose a specific kind of I/O (block, record, text, FLAM4, ...), up to 32 conversions (decoding, decryption, decompression, character set, ...) and a formatting method (binary, text, xml, ...) for the data.

4.3.8 OVERLAY *FIO*

SYNOPSIS

```
HELP:   File IO [BLK]
PATH:   XCNV.INPUT.SAV.FILE
TYPE:   OVERLAY
SYNTAX: FIO.{BLK()/REC()/TXT()/FL4()}
```

DESCRIPTION With the overlay FIO you can choose different kind of file IO methods for read operations.

4.3.9 OBJECT *BLK*

SYNOPSIS

```
HELP:   Read a file block-oriented (mainly for open world)
PATH:   XCNV.INPUT.SAV.FILE.FIO
TYPE:   OBJECT
SYNTAX: BLK(NAME='str'/STREAM/DUMMY,BLKSIZ=num,DATTYP=BINARY/CHAR/TEXT/XML,PRNCTR=DETACH/ ↩
    RETAIN/ERASE,FRCBLK,ADDDLM,SUBSYS(),REMOVE)
```

DESCRIPTION This object defines the block oriented read operation for files. This method reads block size bytes as one chunk and gives this block simply in binary form to the subsequent conversion layer. This is the most famous and simplest way to read a file. This read method is useful for all non record oriented file formats.

You can also read record oriented data sets. In this case the block size defines how many records are processed in one chunk. If the next complete record too big for this block, it is copied as first record of the next block.

If you read records with FIOBLK there are no capability to store any attributes except the record length. Means print control character and slot number are lost. Record oriented data sets you can use FIOREC to manage record attributes separately. Print control character can remain in the record oriented data block or detached, but the attribute is lost after the read operation.

You can also enforce block orientation on a record-oriented device, in which case record boundaries are lost and the block is filled byte-wise. This will result in better performance, but a transparent read of different file formats and normal host data sets is not possible anymore. Additionally, you can also add the system delimiter behind each record to produce a valid block for text formatting.

ARGUMENTS

- STRING:NAME='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- NUMBER:DATTYP=BINARY/CHAR/TEXT/XML -Definition of data type for better handling [NONE]
  - BINARY -Binary data (not printable)
  - CHAR -Character data (no delimiter)
  - TEXT -Text data (with delimiter)
  - XML -XML data (like CHAR)
- SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]
- SWITCH:ADDDLM -Add a delimiter behind each record to produce a text block [FALSE]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.3.10 PARAMETER *BLKSIZ*

SYNOPSIS

```
HELP:   Block size for read operations [AUTO]
PATH:   XCNV.INPUT.SAV.FILE.FIO.BLK
TYPE:   NUMBER
SYNTAX: BLKSIZ=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.3.11 PARAMETER *PRNCTR*

SYNOPSIS

```
HELP:   Print control character handling if read ASA or MCC records [DETACH]
PATH:   XCNV.INPUT.SAV.FILE.FIO.BLK
TYPE:   NUMBER
SYNTAX: PRNCTR=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.3.12 OBJECT *SUBSYS*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   XCNV.INPUT.SAV.FILE.FIO.BLK
TYPE:   OBJECT
SYNTAX: SUBSYS(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
SUBSYS(NAME='name' parm='keyword,keyword=value,value')
SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.3.13 OBJECT *REC*

SYNOPSIS

```
HELP:   Read a file record-oriented (mainly for mainframe world)
PATH:   XCNV.INPUT.SAV.FILE.FIO
TYPE:   OBJECT
SYNTAX: REC(NAME='str'/STREAM/DUMMY,RECMOD=STOP/CUT/WRAP,PRNCTR=DETACH/RETAIN/ERASE,SUPPAD, ↩
    PADCHR='bin'/BINARY-ZERO/ASCII-BLANK/EBCDIC-BLANK/UTF08-BLANK/UTF16BE-BLANK/UTF16LE- ↩
    BLANK/UTF32BE-BLANK/UTF32LE-BLANK,FILORG=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZ=num,RECFMT= ↩
    NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC ↩
    /FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/ ↩
    FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/ ↩
    VKA/VKM/VR/VRA/VRM,RECLEN=num,RECCNT=num,LENFMT.{},CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/ ↩
    EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE,RECDLM='bin'/CR-ASCII ↩
    /LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF-EBCDIC/NL-EBCDIC/CRLF-EBCDIC/CR-UTF08/LF- ↩
    UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE/NL-UTF16BE/CRLF-UTF16BE/CR-UTF16LE/LF- ↩
    UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF-UTF32BE/NL-UTF32BE/CRLF-UTF32BE/CR-UTF32LE ↩
    /LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE,KEYPOS=num,KEYLEN=num,SUBSYS(),REMOVE)
```

DESCRIPTION This object defines the record-oriented read operation for files. This method reads records from a data set. You can define the record length and the behavior (STOP/CUT/WRAP) if the record is longer than the defined record length. You can activate the suppression of padding characters and set the padding character. You can define the file organization, block size, record format and a lot of other parameters (see below).

The record length is always the real data length after all deductions. For example, for a FBA with a LRECL of 121, the record length of FLAM is 120 because the print control character is normally an attribute and not a part of the record.

On record oriented systems (mainly mainframes), a lot of these parameters are known from the catalog and need not be defined. On other platforms (Windows, Unix, . . .), catalogs do not exist. On such systems, you must specify how the record-oriented file was written (record format, length format).

FLAM supports some record formats on mainframes which are not supported by the corresponding catalog system. For example, record formats with print control characters for VSAM data sets are supported by FLAM. If you know your data in a KSDS has a print control character in the first byte, you can choose the corresponding record format (VKA/VKM) to DETACH the print control character as attribute from the record.

The record I/O component also supports text files with delimiters. The record format *text* can be used to parse records form a delimiter-based text file, for which you can define the character set. It is also possible to parse binary data into records based on

a custom delimiter. For this, the record format DLM can be used. The record format BIN simply wraps the data into records of the provided record length. The final record may be shorter. With the FIX record format, the last record also has the provided record length. For variable length record formats (VAR) on non-record-oriented systems, the length format must be defined. All these formats are supported to transfer data between record-oriented and block-oriented systems.

For USS on z/OS the default length format for variable records conforms to SVC99 allocation with FILEDATA=RECORD. The record format TXT works like FILEDATA=TEXT if default settings used, aso.

Attributes (length, slot number of RRDS, print control character) are stored in front of each record on non-record-oriented systems. Example: If you read a RRDS with print control characters (FRA) and you write this on UNIX as record format VRA to a file, the file format looks like this:

```
LLLLnnnnnnnnCdddd..dddd
LLLLnnnnnnnnCdddd..dddd
LLLLnnnnnnnnCdddd..dddd
...
...
...
LLLLnnnnnnnnCdddd..dddd
LLLLnnnnnnnnCdddd..dddd
```

Where LLLL is the record length, nn…nn is the slot number C is the print control character and dd..dd is the record data. To read such a file on a UNIX system, the record format and length format (if you write with defaults, you can read with defaults) must be defined correctly.

ARGUMENTS

- STRING:NAME='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- NUMBER:RECMOD=STOP/CUT/WRAP -Mode used to read records [STOP]
  - STOP -Stop if record too long
  - CUT -Cut if record too long
  - WRAP -Wrap if record too long
- SWITCH:SUPPAD -Suppress trailing padding characters (useful for fixed records)
- STRING:PADCHR='bin'/BINARY-ZERO/ASCII-BLANK/EBCDIC-BLANK/UTF08-BLANK/UTF16BE-BLANK/UTF16LE-BLANK/UTF32BE-BLANK/UTF32LE-BLANK -Define padding character for suppression [automatic]
  - BINARY-ZERO -Padding with 0x00
  - ASCII-BLANK -Padding with 0x20
  - EBCDIC-BLANK -Padding with 0x40
  - UTF08-BLANK -Padding with 0x20
  - UTF16BE-BLANK -Padding with 0x0020
  - UTF16LE-BLANK -Padding with 0x2000
  - UTF32BE-BLANK -Padding with 0x00000020
  - UTF32LE-BLANK -Padding with 0x20000000
- NUMBER:FILORG=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [SEQ]
  - SEQ -Sequential data set [DEFAULT]
  - LIB -Library (ZOS-PDSE)
  - PDS -Partitioned data set (ZOS-PDS)

  - USS -UNIX files (path based)
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)

- NUMBER:RECFMT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format for read operations [NONE]
  - NONE -No format defined
  - BIN -Open -Binary (undefined) data (only the last record are shorter then record length)
  - TXT -Open -Text data with standard delimiter (result is a stripped variable length records)
  - DLM -Open -Records separated by a special delimiter with the result of variable length records
  - VAR -Open -Variable length record (if necessary define length format)
  - VAR-ASA -Open -Variable length record with ASA print control character (if necessary define length format)
  - VAR-MCC -Open -Variable length record with machine print control codes (if necessary define length format)
  - FIX -Open -Fix length record (file size =N * record length)
  - FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
  - FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
  - VAR-REL -Open -Variable relative record (if necessary define length format)
  - VAR-REL-ASA -Open -Variable relative record with ASA print control character (if necessary define length format)
  - VAR-REL-MCC -Open -Variable relative record with machine print control codes (if necessary define length format)
  - FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
  - FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size = N * (record length + 9))
  - FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size = N * (record length + 9))
  - F -Host -Fixed-length, unblocked
  - FA -Host -Fixed-length, unblocked, ASA print control characters
  - FM -Host -Fixed-length, unblocked, machine print control codes
  - FS -Host -Fixed-length, standard
  - FSA -Host -Fixed-length, standard, ASA print control characters
  - FSM -Host -Fixed-length, standard, machine print control codes
  - FB -Host -Fixed-length, blocked
  - FBA -Host -Fixed-length, blocked, ASA print control characters
  - FBM -Host -Fixed-length, blocked, machine print control codes
  - FBS -Host -Fixed-length, blocked, standard
  - FBSA -Host -Fixed-length, blocked, standard, ASA print control characters

  - FBSM -Host -Fixed-length, blocked, standard, machine print control codes
  - FE -Host -Fixed-length, ESDS
  - FEA -Host -Fixed-length, ESDS, ASA print control characters
  - FEM -Host -Fixed-length, ESDS, machine print control codes
  - FK -Host -Fixed-length, KSDS
  - FKA -Host -Fixed-length, KSDS, ASA print control characters
  - FKM -Host -Fixed-length, KSDS, machine print control codes
  - FR -Host -Fixed-length, RRDS
  - FRA -Host -Fixed-length, RRDS, ASA print control characters
  - FRM -Host -Fixed-length, RRDS, machine print control codes
  - U -Host -Undefined-length
  - UA -Host -Undefined-length, ASA print control characters
  - UM -Host -Undefined-length, machine print control codes
  - V -Host -Variable
  - VA -Host -Variable, ASA print control characters
  - VM -Host -Variable, machine print control codes
  - VS -Host -Variable, spanned
  - VSA -Host -Variable, spanned, ASA print control characters
  - VSM -Host -Variable, spanned, machine print control codes
  - VB -Host -Variable, blocked
  - VBA -Host -Variable, blocked, ASA print control characters
  - VBM -Host -Variable, blocked, machine print control codes
  - VBS -Host -Variable, blocked, spanned
  - VBSA -Host -Variable, blocked, spanned, ASA print control characters
  - VBSM -Host -Variable, blocked, spanned, machine print control codes
  - VE -Host -Variable, ESDS
  - VEA -Host -Variable, ESDS, ASA print control characters
  - VEM -Host -Variable, ESDS, machine print control codes
  - VK -Host -Variable, KSDS
  - VKA -Host -Variable, KSDS, ASA print control characters
  - VKM -Host -Variable, KSDS, machine print control codes
  - VR -Host -Variable, RRDS
  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes

- NUMBER:RECLEN=num -Record length for read operations [512]
- NUMBER:RECCNT=num -Amount of records handled in one read operation [128]
- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [auto detection]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)

  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)

- STRING:RECDLM='bin'/CR-ASCII/LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF-EBCDIC/NL-EBCDIC/CRLF-EBCDIC/CR-UTF08/LF-UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE/NL-UTF16BE/CRLF-UTF16BE/CR-UTF16LE/LF-UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF-UTF32BE/NL-UTF32BE/CRLF-UTF32BE/CR-UTF32LE/LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE -Delimiter used to define end of record
  - CR-ASCII -Delimiter:carriage return in ASCII (0x0D)
  - LF-ASCII -Delimiter:line feed in ASCII (0x0A)
  - NL-ASCII -Delimiter:new line in ASCII (0x85)
  - CRLF-ASCII -Delimiter:carriage return line feed in ASCII (0x0D0A)
  - CR-EBCDIC -Delimiter:carriage return in EBCDIC (0x0D)
  - LF-EBCDIC -Delimiter:line feed in EBCDIC (0x25)
  - NL-EBCDIC -Delimiter:new line in EBCDIC (0x15)
  - CRLF-EBCDIC -Delimiter:carriage return line feed in EBCDIC (0x0D25)
  - CR-UTF08 -Delimiter:carriage return in UTF-8 (0x0D)
  - LF-UTF08 -Delimiter:line feed in UTF-8 (0x0A)
  - NL-UTF08 -Delimiter:new line in UTF-8 (0xC285)
  - CRLF-UTF08 -Delimiter:carriage return line feed in UTF-8 (0x0D0A)
  - CR-UTF16BE -Delimiter:carriage return in UTF-16BE (0x000D)
  - LF-UTF16BE -Delimiter:line feed in UTF-16BE (0x000A)
  - NL-UTF16BE -Delimiter:new line in UTF-16BE (0x0085)
  - CRLF-UTF16BE -Delimiter:carriage return line feed in UTF-16BE (0x000D000A)
  - CR-UTF16LE -Delimiter:carriage return in UTF-16LE (0x0D00)
  - LF-UTF16LE -Delimiter:line feed in UTF-16LE (0x0A00)
  - NL-UTF16LE -Delimiter:new line in UTF-16LE (0x8500)
  - CRLF-UTF16LE -Delimiter:carriage return line feed in UTF-16LE (0x0D000A00)
  - CR-UTF32BE -Delimiter:carriage return in UTF-32BE (0x0000000D)
  - LF-UTF32BE -Delimiter:line feed in UTF-32BE (0x0000000A)
  - NL-UTF32BE -Delimiter:new line in UTF-32BE (0x00000085)
  - CRLF-UTF32BE -Delimiter:carriage return line feed in UTF-32BE (0x0000000D0000000A)
  - CR-UTF32LE -Delimiter:carriage return in UTF-32LE (0x0D000000)
  - LF-UTF32LE -Delimiter:line feed in UTF-32LE (0x0A000000)
  - NL-UTF32LE -Delimiter:new line in UTF-32LE (0x85000000)
  - CRLF-UTF32LE -Delimiter:carriage return line feed in UTF-32LE (0x0D0000000A000000)

- NUMBER:KEYPOS=num -Key position (>=1) for index/key sequential data sets [0=NONE]
- NUMBER:KEYLEN=num -Key length (>=1) for index/key sequential data sets [if KEYPOS>0 then 8]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.3.14 PARAMETER *PRNCTR*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   XCNV.INPUT.SAV.FILE.FIO.REC
TYPE:   NUMBER
SYNTAX: PRNCTR=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.3.15 PARAMETER *BLKSIZ*

SYNOPSIS

```
HELP:   Block size for read operations [AUTO]
PATH:   XCNV.INPUT.SAV.FILE.FIO.REC
TYPE:   NUMBER
SYNTAX: BLKSIZ=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.3.16 OVERLAY *LENFMT*

SYNOPSIS

```
HELP:   Format of the length field for open platforms
PATH:   XCNV.INPUT.SAV.FILE.FIO.REC
TYPE:   OVERLAY
SYNTAX: LENFMT.{INTEGER()/STRING()/BCD()/HOST()}
```

DESCRIPTION The overlay can be used to define the format of the length filed on open platforms which don't support record formats by the data management system. For example: If you read records on a mainframe and you want to write the records as records (not as binary or text data) on WINDOWS or UNIX then the data will be prefixed with the record length. The format of this length field can be changed over this union.

This length specification will be only used for variable length records. The default mapping method between record formats will be map each host record format the corresponding open variable format.

If the length format not defined the length field will be written as 32 bit integer in platform dependent byte order. This behavior corresponds to the FILEDATA=RECORD parameter for USS files on z/OS. Means if you write a record oriented file to USS without any format changed over this overlay then you can read the record with another MVS application using FILEDATA=RECORD.

If you choose record format text or binary then this will correspond to FILEDATA=TEXT or FILEDATA=BINARY and the length format specification will be ignored. To enable record format mapping based on the data type you can set the environment variable below:

```
FL_RECORD_FORMAT_MAPPING=DATATYPE
```

If this environment variable set, then the length format is only used if you explicitly set a variable record format of the open systems.

Since version 5.1.8, FLAM can detect the 4 byte length fields (integer or host) inside a data stream. We recommend to use one of these formats to have more comfort when reading.

4.3.17 OBJECT *INTEGER*

SYNOPSIS

```
HELP:   Use integer format for the length field
PATH:   XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT
TYPE:   OBJECT
SYNTAX: INTEGER(ENDIAN=SYSTEM/BIG/LITTLE,WIDTH=U16/U32/U64,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in integer format. You can define the byte order, the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [SYSTEM]
  - SYSTEM -Byte order of the current system
  - BIG -Big endian byte order
  - LITTLE -Little endian byte order
- NUMBER:WIDTH=U16/U32/U64 -Bit width/rate [B32]
  - U16 -16 bit unsigned integer
  - U32 -32 bit unsigned integer
  - U64 -64 bit unsigned integer
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.3.18 OBJECT *STRING*

SYNOPSIS

```
HELP:   Use string format for the length field
PATH:   XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT
TYPE:   OBJECT
SYNTAX: STRING(CHRSET=SYSTEM/ASCII/EBCDIC,COUNT=C04/C05/C06,BASE=B10,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in string format (a character per digit). You can define the character set, the character count, the base and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:CHRSET=SYSTEM/ASCII/EBCDIC -Character set [SYSTEM]
  - SYSTEM -Use system code page
  - ASCII -Use ASCII (UTF-8) code page
  - EBCDIC -Use EBCDIC code page
- NUMBER:COUNT=C04/C05/C06 -Character count [C06]
  - C04 -4 character/digits
  - C05 -5 character/digits
  - C06 -6 character/digits
- NUMBER:BASE=B10 -String base [B10]
  - B10 -Base 10 (decimal)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.3.19 OBJECT *BCD*

SYNOPSIS

```
HELP:   Use BCD format for the length field
PATH:   XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT
TYPE:   OBJECT
SYNTAX: BCD(WIDTH=U16/U24/U32,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in BCD format (4 Bit are one decimal digit). You can define the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:WIDTH=U16/U24/U32 -Bit width/rate [B32]
  - U16 -16 bit unsigned BCD (POV with 4 digits)
  - U24 -24 bit unsigned BCD (POV with 6 digits)
  - U32 -32 bit unsigned BCD (POV with 8 digits)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.3.20 OBJECT *HOST*

SYNOPSIS

```
HELP:   Use host format (LLxx) for the length field
PATH:   XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT
TYPE:   OBJECT
SYNTAX: HOST(ENDIAN=SYSTEM/BIG/LITTLE,EXCLUDE)
```

DESCRIPTION The object defines the 4 byte host specific format (LLxx) of the length field. This format uses the first 16 bits as integer in big endian or little endian byte order to represent the record length including or excluding the record length field itself and the last 16 bits are set to zero when writing and ignored when reading.

For example, a record with 136 bytes content has a default length field (big endian, inclusive) of:

```
x'008D0000'
```

The default conforms to the length fields used by the data management system of MVS.

ARGUMENTS

- NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [BIG ENDIAN] for first 16 Bit
  - SYSTEM -Byte order of the current system
  - BIG -Big endian byte order
  - LITTLE -Little endian byte order
- SWITCH:EXCLUDE -Record length field not part of the length value [OFF]

4.3.21 OBJECT *SUBSYS*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   XCNV.INPUT.SAV.FILE.FIO.REC
TYPE:   OBJECT
SYNTAX: SUBSYS(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
SUBSYS(NAME='name' parm='keyword,keyword=value,value')
SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- STRING:NAME='str' -Name of the subsystem
- STRING:PARM='str' -Parameter list for the subsystem (comma separated)

4.3.22 OBJECT *TXT*

SYNOPSIS

```
HELP:   Read a file text-oriented (mainly for open world)
PATH:   XCNV.INPUT.SAV.FILE.FIO
TYPE:   OBJECT
SYNTAX: TXT(NAME='str'/STREAM/DUMMY,RECMOD=STOP/CUT/WRAP,RECLEN=num,RECCNT=num,PRNOUT=num, ↩
    SUBSYS(),REMOVE)
```

DESCRIPTION This object defines the text-oriented read operation for files. This method reads text records from a file with text delimiters. The data must be a text file in the platform specific character set where each record is terminated with the system specific delimiter. You can define the maximum record length and what to do, if a record is longer than the provided record length.

On read, you can also activate the conversion of the data into a printout-like amount of records, i.e. print control characters are replaced by records so that it looks like the data was printed.

ARGUMENTS

- STRING:NAME='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- NUMBER:RECMOD=STOP/CUT/WRAP -Mode used to read records [STOP]
  - STOP -Stop if record too long
  - CUT -Cut if record too long
  - WRAP -Wrap if record too long
- NUMBER:RECLEN=num -Record (Host) length /Line (Unix/Win) length for read operations [512]
- NUMBER:RECCNT=num -Amount of records handled in one read operation [128]
- NUMBER:PRNOUT=num -Amount of lines per page if print control replacement active [0 = inactive]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.3.23 OBJECT *SUBSYS*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   XCNV.INPUT.SAV.FILE.FIO.TXT
TYPE:   OBJECT
SYNTAX: SUBSYS(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- STRING:NAME='str' -Name of the subsystem
- STRING:PARM='str' -Parameter list for the subsystem (comma separated)

4.3.24 OBJECT *FL4*

SYNOPSIS

```
HELP:   Read a FLAMFILE record-oriented
PATH:   XCNV.INPUT.SAV.FILE.FIO
TYPE:   OBJECT
SYNTAX: FL4(NAME/FLAMFILE='str',MEMBER='str',RECLEN=num,RECCNT=num,TRUCAT,NONFIL,PRNCTR= ↩
    DETACH/RETAIN/ERASE,DEVICE=FILE/USER,PASSWD='bin'/DEFAULT,KMELIB='str',KMEFUC='str', ↩
    KMEPAR='str',EXTCMP='str',EXTDCO='str',SECINF=YES/NO/IGNORE/MEMBER,REMOVE)
```

DESCRIPTION This object defines the FLAM4 read operation for files. This method reads records from a FLAMFILE of version 4 or older. It uses the FLAM4 record interface to read members from a FLAMFILE. The parameters below are explained in detail by the FLAM command of FLCL.

ARGUMENTS

- STRING:NAME/FLAMFILE='str' -Name/URL of FLAMFILE to read [required]
- STRING:MEMBER='str' -Name or index of the member in the FLAMFILE
- NUMBER:RECLEN=num -Maximal record length for read operations [512]
- NUMBER:RECCNT=num -Amount of records handled in one read operation [128]
- SWITCH:TRUCAT -Allow truncation of records [FALSE]
- SWITCH:NONFIL -Switch to replace original file name with FILEnnnn [FALSE]
- NUMBER:DEVICE=FILE/USER -Device used to store the FLAMFILE [FLATFILE]
  - FILE -Write to a file
  - USER -Use user IO
- STRING:PASSWD='bin'/DEFAULT -Passphrase to decrypt the FLAMFILE [”]
  - DEFAULT -FLAM4 default password
- STRING:KMELIB='str' -Library name for FKME [”]
- STRING:KMEFUC='str' -Function name of FKME [”]
- STRING:KMEPAR='str' -Parameter for the call of FKME [”]
- STRING:EXTCMP='str' -Name of user exit for compression
- STRING:EXTDCO='str' -Name of user exit for decompression
- NUMBER:SECINF=YES/NO/IGNORE/MEMBER -Define handling of secure information
  - YES -Create secure information at compression
  - NO -Do not store any additional secure information
  - IGNORE -Ignore any security violations on decompression
  - MEMBER -Only member-specific security informations are verified
- SWITCH:REMOVE -Remove FLAMFILE after successful processing [FALSE]

4.3.25 PARAMETER *PRNCTR*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   XCNV.INPUT.SAV.FILE.FIO.FL4
TYPE:   NUMBER
SYNTAX: PRNCTR=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.3.26 OVERLAY *CNV*

SYNOPSIS

```
HELP:   Conversion [NONE]
PATH:   XCNV.INPUT.SAV.FILE
TYPE:   OVERLAY
SYNTAX: CNV[{BLK()/REC()/GZP()/BZ2()/LXZ()/ZIP()/CHR()/BAS()/HSH()}...]
```

DESCRIPTION With the overlay CNV you can choose the different kind of conversions supported at read operations.

4.3.27 OBJECT *BLK*

SYNOPSIS

```
HELP:   Convert a record list to one block (binary or text) of data
PATH:   XCNV.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: BLK(BUFSIZ=num,METHOD=HOST/BIN/REC/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM ↩
    /ORIGINAL,CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE ↩
    /UTF32BE/UCS4LE/UTF32LE,SUPTWS,RPLTAB/RPLHTB=num,RPLVTB=num,RPLCTR=SPACE/SUBSTITUTE/ ↩
    DELETE)
```

DESCRIPTION The block converter can be used to convert records into data blocks with delimiters. On output, it can also add different length formats in front of the records. This conversion component uses the text, block and binary formatting components and provides all their features.

ARGUMENTS

- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:METHOD=HOST/BIN/REC/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM/ORIGINAL -Method for block/text formatting [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - REC -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
  - ORIGINAL -Adds the original data at the end of a line (only for band FMT)
- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- SWITCH:SUPTWS -Suppress trailing whitespace [FALSE]
- NUMBER:RPLTAB/RPLHTB=num -Replace horizontal tabulators by spaces using this tab width [0 =no replacement]
- NUMBER:RPLVTB=num -Replace vertical tabulators by new lines using this tab width [0 = no replacement]
- NUMBER:RPLCTR=SPACE/SUBSTITUTE/DELETE -Replace remaining control characters [NONE]
  - SPACE -Replace control characters with whitespace character (0x20/0x40)
  - SUBSTITUTE -Replace control characters with substitution character (0x1A/0x3F)
  - DELETE -Remove control characters

4.3.28 OBJECT *REC*

SYNOPSIS

```
HELP:   Convert a block (binary or text) of data into a list of records
PATH:   XCNV.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: REC(METHOD=BIN/DLM/REC/L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX,CHRSET=NONE/ ↩
    SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/ ↩
    UTF32LE,RPLFFD=num,RPLTAB/RPLHTB=num,RPLVTB=num,RPLCTR=SPACE/SUBSTITUTE/DELETE,SUPTWS, ↩
    NELDLM,PADCHR=num,RECLEN=num,BUFSIZ=num,INICNT=num,SKPTXT)
```

DESCRIPTION The record converter can be used to convert data blocks with delimiters or on input with different record length formats into data lists with records. This conversion component uses the text and record formatting component and provides all its features.

Auto detection of 4 byte record length fields is attempted if no method is provided. If the data block contains no record length fields and the switch to skip text formatting is not enabled, then all capabilities of the text formatting component are used to form records.

The SKPTXT flag causes input blocks to be only converted to records if record length fields are detected. This can be useful, for example, for reading ZIP archives containing record-oriented files. PKZIP stores record-oriented files with a 4 byte length field (length of the field itself not included) in front of the record data in little endian (L4X). Another example files stored with FILE-DATA=RECORD (B4X) in USS. When reading such a file, the length fields must be interpreted to re-build the corresponding records. This conversion can be added after any I/O or other conversion step.

ARGUMENTS

- NUMBER:METHOD=BIN/DLM/REC/L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX -Method fo r record/text formatting [DEFAULT]
  - BIN -Wrap data block in records of record length
  - DLM -Parse data block for text delimiter (DEFAULT)
  - REC -Use read record length fields (only useful on mainframes)
  - L4I -Parse data based on 4 byte length fields:Little endian integer, length inclusive
  - L4X -Parse data based on 4 byte length fields:Little endian integer, length exclusi ve (ZIP)
  - B4I -Parse data based on 4 byte length fields:Big endian integer, length inclusive
  - B4X -Parse data based on 4 byte length fields:Big endian integer, length exclusive (USS)
  - S4I -Parse data based on 4 byte length fields:System endian integer, length inclusive
  - S4X -Parse data based on 4 byte length fields:System endian integer, length exclusi ve (VAR)
  - HLI -Parse data based on 4 byte length fields:Little endian short (LLxx), length in clusive
  - HLX -Parse data based on 4 byte length fields:Little endian short (LLxx), length ex clusive
  - HBI -Parse data based on 4 byte length fields:Big endian short (LLxx), length inclu sive (MVS)
  - HBX -Parse data based on 4 byte length fields:Big endian short (LLxx), length exclu sive
  - HSI -Parse data based on 4 byte length fields:System endian short (LLxx), length in clusive
  - HSX -Parse data based on 4 byte length fields:System endian short (LLxx), length ex clusive

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assuming n lines per page [0 =no replacement]
- NUMBER:RPLTAB/RPLHTB=num -Replace horizontal tabulators by spaces using this tab width [0 =no replacement]
- NUMBER:RPLVTB=num -Replace vertical tabulators by new lines using this tab width [0 = no replacement]
- NUMBER:RPLCTR=SPACE/SUBSTITUTE/DELETE -Replace remaining control characters [NONE]
  - SPACE -Replace control characters with whitespace character (0x20/0x40)
  - SUBSTITUTE -Replace control characters with substitution character (0x1A/0x3F)
  - DELETE -Remove control characters
- SWITCH:SUPTWS -Suppress trailing whitespaces [FALSE]
- SWITCH:NELDLM -Activate NEL (0x85) as delimiter for ASCII character sets [FALSE]
- NUMBER:PADCHR=num -Padding character [0x00]
- NUMBER:RECLEN=num -Length used to cut the data block in records [512]
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of records for preallocation [128]
- SWITCH:SKPTXT -Convert only if record length fields found in the data [FALSE]

4.3.29 OBJECT *GZP*

SYNOPSIS

```
HELP:   Read ZLIB/GZIP/RCF1950-2 compressed data
PATH:   XCNV.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: GZP(MEMBER='str',BUFSIZ=num)
```

DESCRIPTION The GZIP converter can be used to compress or decompress data streams.

The result of a compression is a binary block. The result of a decompression can be text, XML, a base encoding or binary data. Record boundaries are lost. Please use delimiters if records must be exchanged.

The GZIP decompression can read all supported formats of zlib and writes RFC1952/51/50-compliant data streams. If a IBM ZEDC acceleration card is available and can be used, the ZEDC is used to save CPU time. In this case, however, the compression does not conform to RFC1951. The compressed data is encoded based on RFC1950 and the file format conforms to RCF1952.

For a checksummed GZIP header, an additional switch must be activated, because this feature is not supported by all GZIP implementations. The GZIP header is used to store meta information about the data, which can be used if FLAM is also used for decompression. This can be very useful to clearly define the character set, CCSID and other information about the data.

If no compression level is set and the data is marked as being without redundancy, then level 0 (copy) is used. Otherwise, the default level 6 is used.

Decompression of concatenated GZIP files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
filename/#memberindex

test.gz/#3
test.gz/:2
```

GZIP data streams are written to ZIP archives as compression mode 8. The GZIP header and trailer are replaced by a ZIP local file header. If a deflated file (compression mode 8) is read from a ZIP archive, it is converted to a GZIP data stream which can be inflated with this component.

ARGUMENTS

- `STRING:MEMBER='str' -Index of the member in the GZIP-File to read`
- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation[65536]`

4.3.30 OBJECT *BZ2*

SYNOPSIS

```
HELP:   Read BZIP2 compressed data
PATH:   XCNV.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: BZ2(MEMBER='str',BUFSIZ=num,SMLSPC)
```

DESCRIPTION The BZIP2 converter can be used to compress or decompress data streams compatible with the BZIP2 utility on UNIX systems.

The result of a compression is a binary block. The result of a decompression can be text, XML, a base encoding or binary data. Record boundaries are lost. Please use delimiters if records must be exchanged.

There is no header to store meta information about the data, so all metadata is lost.

If no compression level is defined, the level used depends on the block size. A block size larger than 900KB results in the best compression level 9, greater than 800KB and smaller than 900KB in level 8, and so on. For blocks smaller than 100KB level 1 is used.

Decompression of concatenated BZIP2 files is also possible. All concatenated files will be decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
filename/#memberindex

test.bz/#3
test.bz/:2
```

BZIP2 data is written to ZIP archives as compression mode 12. The BZIP2 header and trailer are replaced by a ZIP local file header. If a BZIP2 file (compression mode 12) is read from a ZIP archive, it is converted to a BZIP2 data stream and can be decompressed with this component.

ARGUMENTS

- `STRING:MEMBER='str' -Index of the member in the BZIP2-File to read`
- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]`
- `SWITCH:SMLSPC -Use small memory space for decompression [FALSE]`

4.3.31 OBJECT *LXZ*

SYNOPSIS

```
HELP:   Read XZ (LZMA) compressed data
PATH:   XCNV.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: LXZ(MEMBER='str',BUFSIZ=num)
```

DESCRIPTION The LZMA/XZ converter can be used to compress or decompress data streams compatible with the XZ utility on UNIX systems.

The result of a compression is a binary block. The result of a decompression can be text, XML, a base encoding or binary data. Record boundaries are lost. Please use delimiters if records must be exchanged.

There is no header to store meta information about the data, so all metadata is lost.

If no compression level is defined and the data is marked as being without redundancy, then level 0 (copy) is used. In all other cases the default level 6 is used.

Decompression of concatenated XZ files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
   filename/:memberindex
   filename/#memberindex

   test.xz/#3
   test.xz/:2
```

XZ data streams are written to ZIP archives as compression mode 14 (LZMA). The XZ header and trailer are replaced by a ZIP local file header. If an LZMA file (compression mode 14) is read from a ZIP archive, it is converted to an XZ data stream which can be decompressed with this component.

ARGUMENTS

- `STRING:MEMBER='str' -Index of the member in the XT-File to read`
- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]`

4.3.32 OBJECT *ZIP*

SYNOPSIS

```
HELP:   Read compressed data (GZIP, BZIP2, LZMA, ...)
PATH:   XCNV.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: ZIP(MEMBER='str',BUFSIZ=num)
```

DESCRIPTION The ZIP converter can read compressed data streams. The compression algorithm is detected automatically. All compression algorithms (GZIP, BZIP2, XZ,...) are supported which are also available separately on the conversion level of XCNV. This component simply uses the dedicated compression components and adds the auto detection. Please refer to each individual compression component for more details.

This component can be used together with FIO.ZIP() to decompress members which were read from a ZIP archive as FIO.ZIP() does not provide any compression functionality by itself.

```
   inp(sav.fil(fio.zip(name=my.zip member=test.txt) cnv.zip()))
```

ARGUMENTS

- `STRING:MEMBER='str' -Index of the member in the compressed file to read`
- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation[65536]`

4.3.33 OBJECT *CHR*

SYNOPSIS

```
HELP:   Convert character sets (ASCII, EBCDIC, UNICODE)
PATH:   XCNV.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: CHR(BUFSIZ=num,RECCNT=num,METHOD=AUTO/BLOCK/RECORD/KEEP/SKIP,CASMOD=UPPER/LOWER/ ↩
    FOLD/SUPPER/SLOWER/USRTAB,FROM='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,MODE=STOP/IGNORE ↩
    /SUBSTITUTE/IDENTITY,TO='str'/DEFAULT/ASCII/EBCDIC,WRTBOM,HDLBOM/BOM,ENL2LF,ELF2NL, ↩
    SUBCHR[num/SYSTEM...],SYSTAB=ICONV,USRTAB='str'/NPAS/SEPA/DELA/DLAX,REPFIL='str'/STDOUT/ ↩
    STDERR,SKPBIN,SKPEQU)
```

DESCRIPTION This component can be used to convert a text stream to a different character set after reading or before writing. Character conversion requires a source CCSID (FROM) and a target CCSID (TO). Using the keyword MODE, you can define the behavior when invalid or incomplete characters are encountered in the input data. It is possible to stop at, ignore or substitute unsupported characters. For the latter case, you can define a substitution character list (SUBCHR) and/or a user substitution table (USRTAB) and/or use one of the pre-defined system substitution tables (SYSTAB) for transliteration. When using a system table, you can override some of the pre-defined code points using a user table. Each character must be defined as UNICODE point in hexadecimal notation.

Each ignore or substitution process, including transliteration, can be recorded in a report file for review. Moreover, you can convert to upper, lower, special case or use case folding. The component also provides powerful support for byte order marks (BOM) including:

- Byte order change handling at read (for example UTF-16LE in UTF-16BE)
- Byte order marks per stream (if block oriented) or record at write
- Charset detection based on byte order marks

You can differentiate between block- and record-oriented conversion by the keyword METHOD.

- record oriented conversion - throws an error if an incomplete byte sequence occurs at the end of a record
- block oriented conversion - no check for record boundaries, regards all data as one block

If the input text is record-oriented then record orientation will be used by default, and you must explicitly enable block-oriented conversion if you do not intend to check the data at each end of a record. If you choose block orientation, the record lengths table will be lost. Additionally, you can enforce to skip incomplete character sequences at the end of a record to keep the length information.

In case of an error, all position and counter values printed to the error trace contain the current indexes. All indexes start at zero. The first block or record is unit 0.

It supports replacement of EBCDIC new line (0x15) characters by line feed (0x25) at character conversion takes place. When converting to UTF-8/16/32, this will result in the new line characters to be converted to line feeds (0x0A) instead of UTF-8 next line characters (0xC285).

In read mode, the component supports auto-detection of charsets. This can be activated by using the keyword DEFAULT as CCSID. Auto-detection will analyze the first block of input data to determine if the input uses ASCII, EBCDIC or a UTF encoding. The statistic-based computation cannot determine the actual ASCII or EBCDIC character set. Instead, the final CCSID is derived from the language identifier of the environment variable LANG.

Character conversion can skipped or enforced if the CCSIDs are equal or or binary data is detected.

To get a list of supported CCSIDs, use the command below:

```
   flcl INFO GET.CCSID
```

For more information, please have a look at the man pages for each parameter.

ARGUMENTS

- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]`
- `NUMBER:RECCNT=num -Initial amount of records for preallocation [128]`

4.3.34 PARAMETER *METHOD*

SYNOPSIS

```
HELP:   Enforces record or block-oriented conversion [AUTO]
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: METHOD=AUTO/BLOCK/RECORD/KEEP/SKIP
```

DESCRIPTION The method of character conversion defines how the input data fields are handled. A data field can be one block or a number of records dependent from how the data is read and converted before character conversion is done. Is the data field a list of records then the record oriented method shall be used. Is the data field a single block the block oriented conversion shall be used. This means that the METHOD must only be set to enforce record or block orientation against the usual way of processing the data.

- Block Orientation - This means all records are handled as one block and the record length information will go lost. If incomplete characters occur at the end of this block it will be remembered as rest and copied to the next block. Using this method there will be no interruption of the data stream.
- Record Orientation - This method ensures that each record contains only complete character sequences. An incomplete character sequence will only be accepted at the end of the last record and handled also as rest for the record in the next data field. To enforce record orientation processing of one block of data there will be no difference to block orientation conversion, besides for processing the last block.

Thus it makes only sense to switch for a record organized data field to block orientation conversion. But this is very critical, as the length information will go lost. This means that there will be no chance to retrieve the record information, besides the record contains some delimiters which can be used to build a new list of record lengths. So we recommend to use the METHOD switch only if you know what you are doing. In some cases analysis of the error trace might result in the correct method that can be used to convert the text.

If an incomplete byte sequence progresses in the next record, then it can be helpful to enforce block oriented conversion, because the record length conversion can be invalid. On the other side, if the next record starts with a new valid byte sequence, then a skip of the damaged byte sequence in the last record can be helpful.

If SKIP is selected then each incomplete character at the end of a record is recorded in the report file with a special code point 0xFFFFFF including the really skipped byte sequence. If this problem happens in the last block you can enforce record oriented

conversion with SKIP to convert the data and the incomplete rest of the last block. They are written to the report file. Normally the corresponding MODE is used for the final block to prevent a rest.

Please don't mix up SKIP as METHOD for the rest of a record contained in a data field and the IGNORE as MODE for the content of a record as unit of work.

The default method depends on the data. If the data is marked as block-oriented or the record type is text with delimiters, then the method BLOCK is used. In all other cases, record orientation with the method KEEP is used.

SELECTIONS

- `AUTO -Automatic orientation (block or record with keep)`
- `BLOCK -Block-oriented conversion (all data is one chunk)`
- `RECORD -Record-oriented conversion (check record boundaries)`
- `KEEP -Keep (Move a rest from the end to begin of next record)`
- `SKIP -Skip a rest at the end of a record (not recommended)`

4.3.35 PARAMETER *CASMOD*

SYNOPSIS

```
HELP:   Define upper, lower or special case conversion [NONE]
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: CASMOD=UPPER/LOWER/FOLD/SUPPER/SLOWER/USRTAB
```

DESCRIPTION With the case modifier, you can perform conversion to upper or lower case as well as apply case folding and some special case conversions.

Case folding converts all characters to lower case but is more compatible with language specific rules for doing so. For example, this may be useful for case-insensitive string comparison if the strings contain non-Latin characters (code point < 128).

This mapping is done for each code point in front of substitution or transliteration. The respective case is accomplished on the basis of a separate Unicode conversion table in memory. This conversion can be manipulated if user mapping is activated and mapping definitions (⋆) are defined through the provided user table.

The options UPPER and LOWER convert characters without expansion, i.e. returned strings will never grow in length. Note, however, that case folding and the special options (convert e.g. ß to SS) can result in an expansion of data.

All these conversions are pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `UPPER -Change character to upper case`
- `LOWER -Change character to lower case`
- `FOLD -Change character to folded case (for compare)`
- `SUPPER -Change character to special upper case`
- `SLOWER -Change character to special lower case`
- `USRTAB -Change character according to user table`

4.3.36 PARAMETER *FROM*

SYNOPSIS

```
HELP:   Conversion from this CCSID [auto detection]
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   STRING
SYNTAX: FROM='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- DEFAULT –Use default CCSID (auto-detect)
- ASCII –Use default ASCII CCSID (environment)
- EBCDIC –Use default EBCDIC CCSID (environment)
- BOMUTF –Determine the correct UTF CCSID from byte order mark (BOM)
- BOMUCS –Determine the correct UCS CCSID from byte order mark (BOM)

4.3.37 PARAMETER *MODE*

SYNOPSIS

```
HELP:   Define handling of non-convertible characters
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: MODE=STOP/IGNORE/SUBSTITUTE/IDENTITY
```

DESCRIPTION With the MODE selection you can define how an invalid or incomplete character is handled inside of a unit by this component. All possible modes are listed at the end of this chapter.

This MODE is valid for the content of each unit (block or record). At the end of a unit there can be an incomplete character. Such a rest will be handled dependent on the METHOD you choose. The rest of the last unit of a data field will be concatenated to the first unit of the next data field. For all other records in a data field the handling of such a rest will be defined by the respective METHOD.

The SKIP case for METHOD is only relevant for incomplete byte sequences at the end of a unit. The ignore and substitution mode is only effective for all characters in the middle of the unit, besides an incomplete byte sequence occurs in the last unit. Please don't mix up the IGNORE case for MODE with the SKIP case of METHOD.

If a system or user substitution table is used the stop, ignore or substitute are only relevant for characters not managed by the CCSIDs and the substitution tables. This means that the USRTAB and SYSTAB are added to the capabilities of character conversion and only for the rest the chosen MODE is used.

4.3.37.1 CONSTANT *STOP*

SYNOPSIS

```
HELP:   Stop at the first non-convertible character
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR.MODE
TYPE:   NUMBER
SYNTAX: STOP
```

DESCRIPTION Stops conversion at the first incomplete or invalid character which can not be converted. An error message is given with the position of abortion and the value in the input stream. An entry will be (STP()) written to the report file to log this event. To be compatible with other conversion tools, this is the default case, but we recommend to use IGNORE or SUBSTITUTE to prevent any abortion.

indexterm:[Constant STOP}

4.3.37.2 CONSTANT *IGNORE*

SYNOPSIS

```
HELP:   Ignore non-convertible characters
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR.MODE
TYPE:   NUMBER
SYNTAX: IGNORE
```

DESCRIPTION Ignores incomplete or invalid characters and writes a report entry for each character. IGNORE is a substitution by no corresponding character that means no substitution table is required.

indexterm:[Constant IGNORE}

4.3.37.3 CONSTANT *SUBSTITUTE*

SYNOPSIS

```
HELP:   Substitute non-convertible characters
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR.MODE
TYPE:   NUMBER
SYNTAX: SUBSTITUTE
```

DESCRIPTION Substitutes incomplete or invalid character and writes a report entry for each character. For SUBSTITUTE you can define a substitution character list (SUBCHR). If no SUBCHR is provided then this option substitutes invalid or incomplete characters with code point 0x00001A. If a SUBCHR is defined all invalid or incomplete characters will be substituted by the defined list of code points that can be up to 8. If you specify a SUBCHR the substitution mode will be enabled.

indexterm:[Constant SUBSTITUTE}

4.3.37.4 CONSTANT *IDENTITY*

SYNOPSIS

```
HELP:   Copy non-convertible characters (supported only for single byte conversions)
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR.MODE
TYPE:   NUMBER
SYNTAX: IDENTITY
```

DESCRIPTION Copies invalid characters from input to output and writes a report entry for each character. Resulting output could be not printable.

indexterm:[Constant IDENTITY}

4.3.38 PARAMETER *TO*

SYNOPSIS

```
HELP:   Conversion to this CCSID [UTF-8]
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   STRING
SYNTAX: TO='str'/DEFAULT/ASCII/EBCDIC
```

DESCRIPTION The TO code defines the encoding string or CCSID for the output text stream. The default value of this CCSID for read operations is UTF-8. For write operations, the default value is derived from the environment variable LANG. If LANG is not set, the default ASCII or EBCDIC system-codepage is used, depending on the platform.

To get a list of supported encoding strings, please use the command below:

```
   flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs please use:

```
   flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
   CCSID='37'    will be equal to CCSID='IBM037'
   CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit. Optionally, you can use one of the keywords below to use the system-specific default ASCII or EBCDIC CCSID.

SELECTIONS

- DEFAULT –Use default CCSID (UTF-8)
- ASCII –Use default ASCII CCSID (environment)
- EBCDIC –Use default EBCDIC CCSID (environment)

4.3.39 PARAMETER *WRTBOM*

SYNOPSIS

```
HELP:   Write byte order mark in front of the output text
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: WRTBOM
```

DESCRIPTION If the write byte order mark keyword is defined, then the corresponding BOM sign is written at the start of the output stream (block oriented) or at the beginning of each record. If this BOM keyword is not used, then the BOM sign will not be written (this is valid independent of the chosen encoding/CCSID). If a valid BOM sign is located in the middle of the input stream, it is removed from the data. If you define the BOM keyword and the target charset contains only single bytes or no valid BOM sign, then no byte order mark will be written to the output file.

If an invalid BOM sign is recognized, then it will be ignored and causes no entry in the report file. If the byte order changes in the middle of text and the BOM handling switch is not activated, the conversion will stop and an error message is thrown.

By default, the write BOM keyword is set to OFF.

4.3.40 PARAMETER *HDLBOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: HDLBOM/BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.3.41 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.3.42 PARAMETER *ELF2NL*

SYNOPSIS

```
HELP:   Replace EBCDIC line feed (0x25) by new line (0x15) on output
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: ELF2NL
```

DESCRIPTION The replacement of EBCDIC line feed control characters (0x25) by new line characters (0x15) at conversion to EBCDIC can be activated on non EBCDIC (ASCII) systems or deactivated on EBCDIC systems by enabling this switch. This is for example required to prepare text files on open platforms for USS on z/OS. The usual delimiter for text in ASCII and UTF is 0x0A and will normally be translated to 0x25 in EBCDIC. On z/OS(USS), however, 0x15 (NL) is expected.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

To ensure a correctly delimited text record, only a line feed and not a carriage return (0x0D) line feed (0x0A) is used. If text formatting is used with method=NL, this will result in a single LF on distributed systems and the internal conversion will be activated by default to produce a valid text file for z/OS under USS on Windows, Unix and other open platforms.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.3.43 PARAMETER *SUBCHR*

SYNOPSIS

```
HELP:   Default substitution character list [0x1A] (maximal 8 code points)
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: SUBCHR[num/SYSTEM...]
```

DESCRIPTION As substitution character (SUBCHR) you can define a list with up to 8 UNICODE points which are used for the substitution of invalid or incomplete characters if the corresponding code point is not defined in the substitution table (see SUBTAB). If you assign to it the keyword SYSTEM the value 0x1A is written as default substitution character.

If you activate substitution and no SUBCHR is defined, then unsupported characters will be replaced by CODEPOINT 0x00001A.

On the other hand SUBCHR will replace every undefined code point in the substitution table, what is the same behavior when using a USRTAB with corresponding SUBCHR values for every possible code point (000000-200000). If SUBCHR but no MODE is defined the substitution mode will be activated.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `SYSTEM -Use system specific substitution char (0x1A for ASCII/UTF, 0x3F for EBCDIC)`

4.3.44 PARAMETER *SYSTAB*

SYNOPSIS

```
HELP:   Define a preloaded system substitution table
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: SYSTAB=ICONV
```

DESCRIPTION With the system table (SYSTAB) you can load a substitution table with a predefined transliteration schema. Entries in the USRTAB can be used to override predefined transliteration entries or add missing transliteration code points. The options listed at the end of this chapter are available.

If a substitution table is used then the MODE option (STOP/IGNORE/SUBSTITUTE) is only relevant for characters that cannot be converted or substituted.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `ICONV -Preload with ICONV//TRANSLIT substitution table`

4.3.45 PARAMETER *USRTAB*

SYNOPSIS

```
HELP:   Name of the file containing the user conversion table
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   STRING
SYNTAX: USRTAB='str'/NPAS/SEPA/DELA/DLAX
```

DESCRIPTION With this parameter, you can choose a predefined subset or standard manipulation for the system tables via keyword (see `flcl help "xcnv.inp .sav.fil.cnv.chr.usrtab"` for available keywords). Alternatively, you can use a file containing a user defined substitution table (USRTAB).

For the user table file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   USRTAB=NPAS                          ; use XOEV predefined subset
   USRTAB=DELA                          ; German Latin single byte subset
   USRTAB='~.MYUSRTAB(USRTAB1)'         ; PDS on z/OS
   USRTAB='<SYSUID>.MYUSRT02'           ; PS dataset on z/OS
   USRTAB='~/myusrtab3.txt'             ; Unix path name
   USRTAB='<HOME>\usrtab\myusrtab4.txt' ; Windows path name
```

A user defined substitution table has the syntax listed below:

```
   usr_tab_file         -> usr_definition_list
   usr_definition_list  -> usr_definition usr_definition_list
                        |  EMPTY
   usr_definition       -> '(' codepoint_definition ')'
   codepoint_definition -> cp_activation
                        |  cp_deactivation
                        |  cp_transliteration
                        |  cp_standard_mapping
                        |  cp_usr_case_mapping
   cp_activation        -> ['+'] CODEPOINT  [ '-' CODEPOINT ]
   cp_deactivation      ->  '-'  CODEPOINT  [ '-' CODEPOINT ]
   cp_transliteration   -> ['+'] CODEPOINT '=' code_point_list
   cp_standard_mapping  ->  '*'  CODEPOINT '=' code_point_list
   cp_usr_case_mapping  ->  '^'  CODEPOINT '=' code_point_list
   code_point_list      -> CODEPOINT separator code_point_list
                        |  CODEPOINT
                        |  EMPTY
   separator            -> '/'
```

```
                        |  ',' 
                        |  ';'
```

Note: In the description below "/" is used as code point separator.

The optional signs in front of a code point (CP) have the following meaning:

```
  '+' - a valid transliteration (only if CP not in the target set)
  '*' - a valid mapping         (this translation is always done)
  '^' - a case mapping/folding  (only if case=usrtab is activated)
  '-' - an invalid definition   (removes CP for subset definitions)
```

If no sign is used, then a transliteration for a valid code point is defined. Valid code points are accepted, invalid code points result in in an error. Error handling depends on the mode (STOP,IGNORE, SUBSTITUTE). Definition of + code points without code point list, thus without = is equal definition of valid code points. With an asterisk *, you can define an enforced mapping for this code point. For example, this can be used to delete this character if no code point list is provided or to translate this character always in another value. This can also be used to convert code points outside of a subset into this subset.

In contrast to the transliteration (+ is only done if a character doesn't exist in the target encoding), the mapping is always done for each character. Invalid (−) code points can be used to deactivate a code point. To activate (+) or deactivate (−) a range of code points you can use the minus mark (−) followed by one sub code point. The range goes from the smaller code point to the bigger one. For example:

```
  (-00-7F) deactivate all US-ASCII code points
  (+39-30) activate all decimal digits
```

The range operator (−) with the optional plus sign (+) can also be used to activate code points.

To define an own case mapping or folding (required for certain subsets) the caret sign (^) can be used. This mapping is only done if the user table is activated for case mapping (CASE=USRTAB).

To define transliterations/substitutions/mappings an assignment (=) of a code point list is required. The code point list can contain a maximum of 8 code points. If no code point list is specified, then the character is ignored. A transliteration to itself simply activates this code point. The same is valid for mapping definitions. A code point is a hexadecimal number representing a 21 bit UNICODE point.

```
  00000000 to 001FFFFF hex
```

If a code point is not defined in the substitution table (USRTAB and/or SYSTAB), the appropriate substitution character (SUBCHR) is used. If SUBCHR is not set, no substitution is performed. Depending on the MODE, character conversion stops with an error (STOP) or ignores the character (IGNORE).

Text before and after brackets are comments.

Between "(" and "=" or "/" or ")" you can define hex digits until the first non-hex digit. All non-hex digits up to the next separator are interpreted as comment. Leading whitespace is ignored.

```
  REPLACE GERMAN SZ (00DF=0073/0073)                  WITH ss
THIS IS AN EXAMPLE FOR COMMENTS IN CODE POINT LIST:
  REPLACE EURO MARK (20AC= 45#E / 55#U / 52#R / 4F#O) WITH EURO
  REPLACE BOM MARK  (EFFF=)                           WITH NOTHING
```

Leading zeros are allowed. If you don't supply a hex value, then 0x00 is used.

```
  REPLACE GERMAN SZ (00DF=/)                   WITH 0x00 0x00
  REPLACE EURO MARK (20AC=00000045/00055/000052/000004F) WITH EURO
```

ATTENTION: Please don't use parentheses "()" or the operators in your comments.

To describe your own subsets, you can use a user table without a system table. The USRTAB can also be used to overwrite or add transliterations when a system substitution table (for example SYSTAB=ICONV) is used. The transliteration works recursively, that is if one of the substitution code points is not in the target set, this substitution will be used instead and so on.

```
REPLACE GERMAN OE (0000D6=004F/0045)       WITH O E
REPLACE GERMAN SZ (0000DF=0073/0073)       WITH s s
REPLACE EURO MARK (0020AC=00D6/00DF/52/4F) WITH OE SZ R O
```

If you have a EURO sign in your text and convert it to *Latin1*, the resulting byte string will be *D6DF524F*. On the other hand, if you convert it to *ASCII*, the byte string will be *4F457373524F*.

The mapping as described above is always done and is also done recursive including the transliteration result. For example, if you define a mapping `(*30=39)` then the target data won't have any zero in the resulting text.

By replacing other code points recursively, you could easily cause infinite loops. To prevent this, the amount of replacements and the length of one replacement is limited to a maximum of 64. Note that this could still result in a large expansion of data. Therefore, be careful about defining recursive substitutions.

A sample user table for the ICONV system table to change the transliteration of German umlauts to AE, OE or UE is located in the SAMPLE directory under CCUTDEXL. Another sample user table called CCUTNPAS defines the *string.latin* subset (XOEF) which is mainly used for statutory reporting.

The order of definition is the order of processing the definitions. For exampel, if you define a mapping or a transliteration for code point X and later you make this code point invalid, then the code point is invalid. On the other hand, you can first deactivate all code points, then activate your subset and define your transliterations (best fit) or mappings.

For example: To define a non-expansion best fit mapping a user table can be used to delete all combined characters for single byte code pages. For the ICONV system transliteration table, a sample user table can be found under the name CCUTBFM1.

Some other subset user table definitions were added. For example:

- currently valid SEPA characters (<128) for money transfers (CCUTSEPA)
- characters valid in ISO-8859-15, CP1252 and IBM1142 (CCUTDELA)
- characters valid in ISO-8859-15, CP1252, IBM1142 and XOEF (CCUTDLAX)

All user table definitions are pre-calculated once at the beginning of execution to reduce the required CPU time for the conversion. So, the use of a user table increases the cost to open a file but has no effect on CPU utilization when converting the data.

To minimize the effort and to simplify the usage of user table, the most common subsets can be selected by keyword. In this case, a pre-calculated load module (DLL) is used. The subset definition can be for UTF (21 Bit UNICODE) or UCS (16 Bit UNICODE). If it is a UCS subset, then no valid code point is greater than 0xFFFF and you must use UCS CCSIDS to ensure that no code point greater than 0xFFFF is accepted.

If you have a working user table definition for a popular subset or system table manipulation, you can request a keyword for it. Please report this requirement over our issue tracking system

```
http://www.flam.de/en/technology/support/issues/
```

and attach the corresponding user table definition.

SELECTIONS

- `NPAS -UCS subset for statutory reporting (New passport, XOEF, String.Latin)`
- `SEPA -UCS subset of valid SEPA character smaller then 128`
- `DELA -UCS subset for German Latin of IBM1141, ISO8859-15 and CP1252`
- `DLAX -UCS subset for German Latin of IBM1141, ISO8859-15, CP1252 and XOEF`

4.3.46 PARAMETER *REPFIL*

SYNOPSIS

```
HELP:   Name of the report file to log substitutions
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   STRING
SYNTAX: REPFIL='str'/STDOUT/STDERR
```

DESCRIPTION With the report capability, you can log every substitution (stop / ignore / substitute / translit / skip / ...) to a log file. For the report file, all rules and replacements are valid, which are described under "FILENAME HANDLING" above. For example:

```
   REPORT=STDERR                          ; is SYSPRINT on z/OS
   REPORT='~.MYREPORT(REPORT1)'           ; PDS on z/OS
   REPORT='<SYSUID>.MYREPRT2'             ; PS dataset on z/OS
   REPORT='~/myreport3.txt'               ; Unix path name
   REPORT='<HOME>\reports\myreport4.txt'  ; Windows path name
```

A substitution report has the syntax listed below:

```
   report_file         -> sub_definition_list
   sub_definition_list -> sub_definition sub_definition_list
                       |  EMPTY
   sub_definition      -> SUB '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string '='
                             code_point_list byte_string ')'
                       -> XLT '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string '='
                             code_point_list byte_string ')'
                       |  IGN '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string ')'
                       |  STP '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string ')'
                       |  IDT '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string ')'
                       |  SKP '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string ')'
                       |  UKW '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string ')'
   code_point_list     -> CODEPOINT '/' code_point_list
                       |  EMPTY
   code_point          -> CODEPOINT
                       |  EMPTY
   byte_string         -> '(' BYTESTRING ')'
                       |  EMPTY
```

A comment behind brackets explains the reason for substitution:

```
'invalid character'
'incomplete character'
'incomplete character at the end of a record'
```

The notations STP (stop), IGN (ignore), SUB (substitute), XLT (transliteration), IDT (identical) or SKP (skip) before round brackets indicate the action taken. If the task can not be determined then UKW (unknown) is used. A ignore of an invalid character is realized as substitution with nothing. All substitutions with a length of 0 are indicated with IGN (ignore). The position of such an abnormal/irregular character is coded inside the braces:

- UNIT number of records or blocks (depends on the data)
- OFFSET byte offset up to this unit in the complete data stream

- POSITION number of bytes inside the unit where the activity was done

All counting starts with 0. The first block or record has unit 0.

The position is relative to the data stream which was presented to the character conversion module. After conversions and formatting the position values can be far off from the input file which was read originally. For example, when reading a text file with text formatting to records, then delimiting characters are not included in the byte offset, the units are records and the position indicates which character was the issue for this log entry.

The CODEPOINTs (UNICODE) are taken from the defined substitution table The BYTESTRINGs are taken from the input (read characters) and output data (written characters). BYTESTRINGs and CODEPOINTs are provided if they are known in the cases of such an irregularity. In some cases, these values can be missing.

For interpreting the report file, you must know that only the values inside round brackets are meaningful. Text before or behind round brackets are comments, but this may be subject to changes.

SELECTIONS

- `STDOUT -Write output to stdout`
- `STDERR -Write output to stderr`

4.3.47 PARAMETER *SKPBIN*

SYNOPSIS

```
HELP:   Skip character conversion if binary data is detected [FALSE]
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: SKPBIN
```

DESCRIPTION If set, this switch bypasses the character conversion module if the from-code cannot be detected. Charset detection is performed only when no CCSID or DEFAULT is provided. For automated conversion, it is useful to activate this mechanism. This may allow you to process binary and character data using the same command by performing character conversion only on printable, but not on binary data.

4.3.48 PARAMETER *SKPEQU*

SYNOPSIS

```
HELP:   Skip character conversion if from-code equals to-code [FALSE]
PATH:   XCNV.INPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: SKPEQU
```

DESCRIPTION If set, this switch bypasses the character conversion module if the from-code equals the to-code (which may happen when using charset auto detection), resulting in increased performance. This, however, also means that the input data will not be checked for invalid characters.

4.3.49 OBJECT *BAS*

SYNOPSIS

```
HELP:   Convert Base16/32/64/85 encoded data to binary data
PATH:   XCNV.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: BAS(BUFSIZ=num,RECCNT=num,METHOD=BASE16/BASE32/BASE64,MODE=BLOCK/RECORD,CHRSET=NONE ↩
    /SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/ ↩
    UTF32LE,CRCCHK,ARMCHK,RSTCHK,SKPNOB)
```

DESCRIPTION The base converter can be used to convert data from binary format to printable text and vice versa using one of a couple of base encoding schemes (RFC 4648). When writing, binary data is encoded, when reading, encoded text is decoded. Encoding data to a different base results in expansion of the data. The ratio of expansion depends on the base used.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding and decoding can be performed in one of two modes:

The block mode converts the input data as one large block of binary data. Padding will only be added at the end of the output file. This mode is the default for any type of input data, unless specified otherwise. This implies that record-based data will not retain record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* length parameter specifies after how many characters a line is wrapped. The line delimiter can be configured, but can also be omitted, which is useful to write the output record-oriented. When writing record-oriented, each output line results in one record. Note that if the *line* length parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

The record mode can be useful if the input data is record-based. In record mode, each record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. To read a file that has been encoded in record mode, setting mode=record is mandatory. Otherwise, the result might be garbage.

The character set used (ASCII or EBCDIC) can also be configured. By default, the system-specific character set is used.

Mainly for PGP support, the component supports optional ASCII armor (also in EBCDIC) headers and trailers around the base encoded data (see RFC4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified. When reading, the trailer string can optionally be verified to match the header string.

Another feature for OpenPGP support is a CRC checksum that is appended to the data. This is also optional and usable independently of the data, the encoding schema and the ASCII armor support. When writing, a switch can be enabled to add a CRC24 checksum conforming to RCF4880 for Base64 encodings. The other encoding schemas use the same mechanism to append a checksum to the base encoded data, but use another CRC variant. Base16 uses CRC32 and Base32 uses CRC40, known from the GSM network. When reading, a switch enables verification of this checksum. By default, no CRC verification is done. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against. If the verification fails, you get an error. If the verification is successful, you get an informational. If there is no checksum appended for verification, you get a warning. The same is true if verification is not requested, but a checksum is found.

ARGUMENTS

- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]`
- `NUMBER:RECCNT=num -Initial amount of records for preallocation [128]`
- `NUMBER:METHOD=BASE16/BASE32/BASE64 -Base encoding selector [AUTO]`
  - `BASE16 -Base16 (hexadecimal) encoding`
  - `BASE32 -Base32 encoding`
  - `BASE64 -Base64 encoding`
- `NUMBER:MODE=BLOCK/RECORD -Enforces record or block-oriented conversion [BLOCK]`
  - `BLOCK -Block-oriented conversion (all data is one chunk)`
  - `RECORD -Record-oriented conversion (convert each record individually)`
- `NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]`
  - `NONE -No character set defined`
  - `SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)`

  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- SWITCH:CRCCHK -Check CRC if available [FALSE]
- SWITCH:ARMCHK -Check ARMOR trailer if available [FALSE]
- SWITCH:RSTCHK -Check if a rest after encoded data [FALSE]
- SWITCH:SKPNOB -Skip base decoding if no base encoding detected [FALSE]

4.3.50 OBJECT *HSH*

SYNOPSIS

```
HELP:   Generation or verification of one way hash values
PATH:   XCNV.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: HSH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/ ↩
    CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)

  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.3.51 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   XCNV.INPUT.SAV.FILE.CNV.HSH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.3.52 OVERLAY *FMT*

SYNOPSIS

```
HELP:   Formatting [BIN]
PATH:   XCNV.INPUT.SAV.FILE
TYPE:   OVERLAY
SYNTAX: FMT.{BIN()/BLK()/REC()/TXT()/XML()}
```

DESCRIPTION With the overlay FMT you can choose the different kind of formatting methods to convert the data lists in elements at read operations.

4.3.53 OBJECT *BIN*

SYNOPSIS

```
HELP:   Format data in binary elements (block/records)
PATH:   XCNV.INPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: BIN(BUFSIZ=num,RECCNT=num,TERASE,SEGINT=NONE/CHKS/AUTH/MAC,LENMOD=NONE/FAST/DYNAMIC ↩
    /COMPACT,LENCNF=NONE/VEIL/ENC,DATMOD=NONE/FAST/DYNAMIC/COMPACT,DATCNF=NONE/VEIL/ENC)
```

DESCRIPTION The object "format binary" converts each item from a data list (blocks or records with attributes) to an element. The data is untouched, the assigned element's types depend on the attributes and the attributes are stored for each element. This formatting method is mainly useful for records or binary chunks of data.

ARGUMENTS

- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:RECCNT=num -Initial amount of records for preallocation [128]
- SWITCH:TERASE -Erase data from memory when no longer needed [FALSE]
- NUMBER:SEGINT=NONE/CHKS/AUTH/MAC -Segment integrity mode [NONE]
  - NONE -No integrity protection
  - CHKS -Integrity protection with checksum
  - AUTH -Integrity protection with encrypted checksum
  - MAC -Integrity protection with message authentication codes
- NUMBER:LENMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for length fields [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:LENCNF=NONE/VEIL/ENC -Encryption mode for length fields [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption
- NUMBER:DATMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for data [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)

  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:DATCNF=NONE/VEIL/ENC -Encryption mode for data [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption

4.3.54 OBJECT *BLK*

SYNOPSIS

```
HELP:   Format data in one block element (all meta data lost)
PATH:   XCNV.INPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: BLK(BUFSIZ=num,TERASE,SEGINT=NONE/CHKS/AUTH/MAC,LENMOD=NONE/FAST/DYNAMIC/COMPACT, ↩
    LENCNF=NONE/VEIL/ENC,DATMOD=NONE/FAST/DYNAMIC/COMPACT,DATCNF=NONE/VEIL/ENC)
```

DESCRIPTION The object "format block" formats all data items (a block or a bunch of records) as an element containing the raw block/chunk of data. All additional attributes and record boundaries are lost.

ARGUMENTS

- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- SWITCH:TERASE -Erase data from memory when no longer needed [FALSE]
- NUMBER:SEGINT=NONE/CHKS/AUTH/MAC -Segment integrity mode [NONE]
  - NONE -No integrity protection
  - CHKS -Integrity protection with checksum
  - AUTH -Integrity protection with encrypted checksum
  - MAC -Integrity protection with message authentication codes
- NUMBER:LENMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for length fields [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:LENCNF=NONE/VEIL/ENC -Encryption mode for length fields [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption
- NUMBER:DATMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for data [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:DATCNF=NONE/VEIL/ENC -Encryption mode for data [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption

4.3.55 OBJECT *REC*

SYNOPSIS

```
HELP:   Format data in several wrapped record elements (all meta data lost)
PATH:   XCNV.INPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: REC(METHOD=L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX,WRPERR,RECLEN=num,BUFSIZ ↩
    =num,INICNT=num,TERASE,SEGINT=NONE/CHKS/AUTH/MAC,LENMOD=NONE/FAST/DYNAMIC/COMPACT,LENCNF ↩
    =NONE/VEIL/ENC,DATMOD=NONE/FAST/DYNAMIC/COMPACT,DATCNF=NONE/VEIL/ENC)
```

DESCRIPTION By default (i.e. no method provided), the object "format record" formats data blocks to records by slicing the block into records of the provided record length if none of the valid 4 byte length formats is detected.If the data is already organized as records, then the data is re-wrapped. You can enforce an error to prevent the default wrapping if no record length information is found.

ARGUMENTS

- NUMBER:METHOD=L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX -Method for record for matting [AUTO]
  - L4I -Parse data based on 4 byte length fields:Little endian integer, length inclusive
  - L4X -Parse data based on 4 byte length fields:Little endian integer, length exclusive (ZIP)
  - B4I -Parse data based on 4 byte length fields:Big endian integer, length inclusive
  - B4X -Parse data based on 4 byte length fields:Big endian integer, length exclusive (USS)
  - S4I -Parse data based on 4 byte length fields:System endian integer, length inclusive
  - S4X -Parse data based on 4 byte length fields:System endian integer, length exclusive (VAR)
  - HLI -Parse data based on 4 byte length fields:Little endian short (LLxx), length inclusive
  - HLX -Parse data based on 4 byte length fields:Little endian short (LLxx), length exclusive
  - HBI -Parse data based on 4 byte length fields:Big endian short (LLxx), length inclusive (MVS)
  - HBX -Parse data based on 4 byte length fields:Big endian short (LLxx), length exclusive
  - HSI -Parse data based on 4 byte length fields:System endian short (LLxx), length inclusive
  - HSX -Parse data based on 4 byte length fields:System endian short (LLxx), length exclusive
- SWITCH:WRPERR -If no method defined (auto detection) enforce an error if no length format (wrapping) detected [FALSE]
- NUMBER:RECLEN=num -Maximum length of a record [512]
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of records for preallocation [128]
- SWITCH:TERASE -Erase data from memory when no longer needed [FALSE]
- NUMBER:SEGINT=NONE/CHKS/AUTH/MAC -Segment integrity mode [NONE]
  - NONE -No integrity protection

  - CHKS -Integrity protection with checksum
  - AUTH -Integrity protection with encrypted checksum
  - MAC -Integrity protection with message authentication codes
- NUMBER:LENMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for length fields [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:LENCNF=NONE/VEIL/ENC -Encryption mode for length fields [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption
- NUMBER:DATMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for data [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:DATCNF=NONE/VEIL/ENC -Encryption mode for data [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption

4.3.56 OBJECT *TXT*

SYNOPSIS

```
HELP:   Format data based on delimiters in text elements (record and rest)
PATH:   XCNV.INPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: TXT(METHOD=BIN/DLM/REC,CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/ ↩
    UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE,RPLFFD=num,SUPTWS,NELDLM,PADCHR=num,RECLEN= ↩
    num,BUFSIZ=num,INICNT=num,BINERR,CHRERR,TERASE,SEGINT=NONE/CHKS/AUTH/MAC,LENMOD=NONE/ ↩
    FAST/DYNAMIC/COMPACT,LENCNF=NONE/VEIL/ENC,DATMOD=NONE/FAST/DYNAMIC/COMPACT,DATCNF=NONE/ ↩
    VEIL/ENC)
```

DESCRIPTION The object "format text" splits the data into text records and rest elements based on delimiters (METHOD=DLM). The rest element contains the delimiter and optional trailing whitespace if the suppression of trailing whitespace is activated. The text record and the rest element together compose the original data.

If the data contains no delimiters but records length are available, record formatting (METHOD=REC) is used. In this case, the rest element is empty unless suppression of trailing whitespace is used. If no delimiters and no length fields are found and the BINERR switch is not set, then binary formatting (wrapping of the data into records of the defined length) is done. If the BINERR switch is set, an error is triggered if the data is binary. Delimiter, record or binary formatting can be enforced by defining the corresponding method or the data dependent automatism can be used.

If you have text in one of the ISO code pages with 0x85 (NEL) as delimiter, you can optionally enable recognition of NEL as delimiter. NEL (next line) is optional because this value is the general currency sign or the continuation symbol and not a new line in many single byte ASCII codepages.

- 0x00 (padding character (can be changed))
- 0x0A (carriage return (EBCDIC 0x25))
- 0x0D (line feed (EBCDIC(0x0D)))
- 0x0A, 0x0D (carriage return line feed)
- 0x0A, . . . , 0x0A, 0x0D (dirty delimiters)
- 0x0A, 0x0D, . . . , 0x0D (dirty delimiters)
- 0x0C (form feed (EBCDIC 0x22))
- 0x85 (new/next line (EBCDIC 0x15, optional if single byte ASCII))

Text formatting supports a lot of powerful features which can be accessed with the parameters below.

ARGUMENTS

- NUMBER:METHOD=BIN/DLM/REC -Method for text formatting [DEFAULT]
  - BIN -Wrap data block in records of record length
  - DLM -Parse data block for text delimiter (DEFAULT)
  - REC -Parse data block based on record length fields
- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [auto detection]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assuming n lines per page [0 =no replacement]
- SWITCH:SUPTWS -Suppress trailing whitespaces [FALSE]
- SWITCH:NELDLM -Activate NEL (0x85) as delimiter for ASCII character sets [FALSE]
- NUMBER:PADCHR=num -Padding character (is regarded as additional delimiter) [0x00]
- NUMBER:RECLEN=num -Maximum length of a text record (Host) /line (Unix/Win) [512]
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of records for preallocation [128]

- SWITCH:BINERR –Enforce an error if binary data detected [FALSE]
- SWITCH:CHRERR –Enforce an error if character data without delimiter detected [FALSE]
- SWITCH:TERASE –Erase data from memory when no longer needed [FALSE]
- NUMBER:SEGINT=NONE/CHKS/AUTH/MAC –Segment integrity mode [NONE]
  - NONE –No integrity protection
  - CHKS –Integrity protection with checksum
  - AUTH –Integrity protection with encrypted checksum
  - MAC –Integrity protection with message authentication codes
- NUMBER:LENMOD=NONE/FAST/DYNAMIC/COMPACT –Compression mode for length fields [COPY]
  - NONE –No compression =copy of data
  - FAST –Fast compression (more data, less CPU)
  - DYNAMIC –Dynamic compression (best in data and CPU)
  - COMPACT –Compact compression (less data, more CPU)
- NUMBER:LENCNF=NONE/VEIL/ENC –Encryption mode for length fields [NONE]
  - NONE –No confidentiality protection
  - VEIL –Confidentiality protection with veiling
  - ENC –Confidentiality protection with encryption
- NUMBER:DATMOD=NONE/FAST/DYNAMIC/COMPACT –Compression mode for data [COPY]
  - NONE –No compression =copy of data
  - FAST –Fast compression (more data, less CPU)
  - DYNAMIC –Dynamic compression (best in data and CPU)
  - COMPACT –Compact compression (less data, more CPU)
- NUMBER:DATCNF=NONE/VEIL/ENC –Encryption mode for data [NONE]
  - NONE –No confidentiality protection
  - VEIL –Confidentiality protection with veiling
  - ENC –Confidentiality protection with encryption

4.3.57 OBJECT *XML*

SYNOPSIS

```
HELP:   Format data stream in XML elements (tags, attributes, data, ...)
PATH:   XCNV.INPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: XML(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/ ↩
    UTF32BE/UCS4LE/UTF32LE,FMTERR,NOCMNT,BUFSIZ=num,INICNT=num,TERASE,SEGINT=NONE/CHKS/AUTH/ ↩
    MAC,LENMOD=NONE/FAST/DYNAMIC/COMPACT,LENCNF=NONE/VEIL/ENC,DATMOD=NONE/FAST/DYNAMIC/ ↩
    COMPACT,DATCNF=NONE/VEIL/ENC)
```

DESCRIPTION The object "format XML" parses block of text data containing an XML document. The XML data is parsed using the Expat library and transformed into FLAM elements, which allows various formatting options when writing, including minimizing and pretty printing the XML data.

The text data must be in UTF-8 or ASCII. Lines must be delimited with 0x0A, 0x0D or 0x0D0A. During parsing, all line delimiters are normalized to line feed characters (0xA) as defined by the XML specification.

XML formatting supports a lot of powerful features which can be accessed with the parameters below.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [auto]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- SWITCH:FMTERR -Enforce a format error if data contains only space or is empty [FALSE]
- SWITCH:NOCMNT -Ignore XML comments [FALSE]
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of records for preallocation [128]
- SWITCH:TERASE -Erase data from memory when no longer needed [FALSE]
- NUMBER:SEGINT=NONE/CHKS/AUTH/MAC -Segment integrity mode [NONE]
  - NONE -No integrity protection
  - CHKS -Integrity protection with checksum
  - AUTH -Integrity protection with encrypted checksum
  - MAC -Integrity protection with message authentication codes
- NUMBER:LENMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for length fields [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:LENCNF=NONE/VEIL/ENC -Encryption mode for length fields [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption
- NUMBER:DATMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for data [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:DATCNF=NONE/VEIL/ENC -Encryption mode for data [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption

4.3.58 OBJECT *ENV*

SYNOPSIS

```
HELP:   Environment
PATH:   XCNV.INPUT.SAV.FILE
TYPE:   OBJECT
SYNTAX: ENV[(KEYWORD='str',VALUE='str')...]
```

DESCRIPTION With the object ENV you can overrule up to 32 environment variables for read operation. This is useful to define (for example) another LANG variable to emulate another platform for character conversion.

ARGUMENTS

- `STRING:KEYWORD='str' -Keyword of environment variable`
- `STRING:VALUE='str' -Value of environment variable`

4.3.59 OBJECT *OUTPUT*

SYNOPSIS

```
HELP:   Outbound parameter
PATH:   XCNV
TYPE:   OBJECT
SYNTAX: OUTPUT(NET.{},SAV.{})
```

DESCRIPTION Over the object OUTPUT you can define how the data is written.

4.3.60 OVERLAY *NET*

SYNOPSIS

```
HELP:   Network [NONE]
PATH:   XCNV.OUTPUT
TYPE:   OVERLAY
SYNTAX: NET.{IPN()/IPS()/MQS()}
```

DESCRIPTION With the overlay NET you can choose the protocol for the remote read or write access. Below You can find the currently supported protocols and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.

4.3.61 OBJECT *IPN*

SYNOPSIS

```
HELP:   IP-Native
PATH:   XCNV.OUTPUT.NET
TYPE:   OBJECT
SYNTAX: IPN(HOST='str',PORT='str',USER='str',AUTH='str')
```

DESCRIPTION The IP native (IPN) method supports IPv4 and IPv6 network communication for remote access to files, archives and other sources and targets of the FLAM infrastructure. For this client-server architecture, the FLIES server must be started and running in the IPN mode listening on a dedicated port (17996) on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or DNS name) and the corresponding service port (number or name) must be specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES server is required to handle requests.

ARGUMENTS

- `STRING:HOST='str' -IP address [localhost]`
- `STRING:PORT='str' -Port number [17996]`
- `STRING:USER='str' -User name [optional]`
- `STRING:AUTH='str' -Authentication data [optional]`

4.3.62 OBJECT *IPS*

SYNOPSIS

```
HELP:   IP-Secure (SSL/TLS)
PATH:   XCNV.OUTPUT.NET
TYPE:   OBJECT
SYNTAX: IPS(HOST='str',PORT='str',USER='str',AUTH='str')
```

DESCRIPTION The IP secure (IPS) method supports IPv4 and IPv6 network communication with SSL/TLS for remote access to files, archives and other sources and targets of the FLAM infrastructure. For this client-server architecture, the FLIES server must be started and running in the IPS mode listening on a dedicated port (17996) on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or DNS name) and the corresponding service port (number or name) must be specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES server is required to handle requests.

ARGUMENTS

- `STRING:HOST='str' -IP address [localhost]`
- `STRING:PORT='str' -Port number [17996]`
- `STRING:USER='str' -User name [optional]`
- `STRING:AUTH='str' -Authentication data [optional]`

4.3.63 OBJECT *MQS*

SYNOPSIS

```
HELP:   MQ-Series
PATH:   XCNV.OUTPUT.NET
TYPE:   OBJECT
SYNTAX: MQS(QMANAGER='str',QUEUE='str',USER='str',AUTH='str')
```

DESCRIPTION The MQ-Series (MQS) method supports IBM® message queuing for remote access to files, archives and other sources and targets of the FLAM infrastructure. For this client-server architecture, the FLIES server must be started and running in MQS mode listening on a dedicated queue on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES server is required to handle requests.

ARGUMENTS

- `STRING:QMANAGER='str' -Queue manager name [""]`
- `STRING:QUEUE='str' -Queue name [""]`
- `STRING:USER='str' -User name [optional]`
- `STRING:AUTH='str' -Authentication data [optional]`

4.3.64 OVERLAY *SAV*

SYNOPSIS

```
HELP:   Storage [FILE]
PATH:   XCNV.OUTPUT
TYPE:   OVERLAY
SYNTAX: SAV.{FILE()}
```

DESCRIPTION With the overlay SAV you can choose the kind of source or target for the read or write operation. Below You can find the currently supported procedures and the corresponding parameter explained in a dedicated chapter.

4.3.65 OBJECT *FILE*

SYNOPSIS

```
HELP:   File handling
PATH:   XCNV.OUTPUT.SAV
TYPE:   OBJECT
SYNTAX: FILE(FMT.{},CNV[{}...],FIO.{},ENV[()...])
```

DESCRIPTION Over the object FILE you can define how a neutral list of FLAM5 elements with type, length, value and attributes is written to a file. For this you can first define a method to format the data (binary, text, XML, ...), up to 32 conversions (character set, compression, encryption, encoding, ...) and a dedicated kind of I/O procedure (block, record, text FLAM4, ...).

4.3.66 OVERLAY *FMT*

SYNOPSIS

```
HELP:   Formatting [BIN]
PATH:   XCNV.OUTPUT.SAV.FILE
TYPE:   OVERLAY
SYNTAX: FMT.{BIN()/BLK()/REC()/TXT()/XML()}
```

DESCRIPTION With the overlay FMT you can choose the different kind of formatting methods to convert elements in data lists at write operations.

4.3.67 OBJECT *BIN*

SYNOPSIS

```
HELP:   Format elements as binary data (block/records)
PATH:   XCNV.OUTPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: BIN(BUFSIZ=num,RECCNT=num)
```

DESCRIPTION The object "format binary" simply concatenates the data of each element to a binary stream. If the elements are records, then the record length and other attributes are still available in the data list.

ARGUMENTS

- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:RECCNT=num -Initial amount of records for preallocation [128]

4.3.68 OBJECT *BLK*

SYNOPSIS

```
HELP:   Format all elements to one block of data (all meta data lost)
PATH:   XCNV.OUTPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: BLK(METHOD=L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX,BUFSIZ=num)
```

DESCRIPTION By default (i.e. no method provided), the object "format block" simply concatenates the data of each element to one block of data. Through the method parameter, you can format blocks of data with 4 byte length fields in front of the element data. All attributes and additional element information is lost. If a length field is added for each element, the data will be organized as binary blocks and it is not possible, for example, to do character conversion or other text oriented things anymore.

ARGUMENTS

- NUMBER:METHOD=L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX -Method for block formatting [JOIN ELEMENTS TOGETHER]
  - L4I -Adds 4 byte length fields:Little endian integer, length inclusive
  - L4X -Adds 4 byte length fields:Little endian integer, length exclusive (ZIP)
  - B4I -Adds 4 byte length fields:Big endian integer, length inclusive
  - B4X -Adds 4 byte length fields:Big endian integer, length exclusive (USS)
  - S4I -Adds 4 byte length fields:System endian integer, length inclusive
  - S4X -Adds 4 byte length fields:System endian integer, length exclusive (VAR)
  - HLI -Adds 4 byte length fields:Little endian short (LL00), length inclusive
  - HLX -Adds 4 byte length fields:Little endian short (LL00), length exclusive
  - HBI -Adds 4 byte length fields:Big endian short (LL00), length inclusive (MVS)
  - HBX -Adds 4 byte length fields:Big endian short (LL00), length exclusive
  - HSI -Adds 4 byte length fields:System endian short (LL00), length inclusive
  - HSX -Adds 4 byte length fields:System endian short (LL00), length exclusive
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]

4.3.69 OBJECT *REC*

SYNOPSIS

```
HELP:   Format each element to one standard record (possible meta data lost)
PATH:   XCNV.OUTPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: REC(BUFSIZ=num,INICNT=num)
```

DESCRIPTION The object "format record" simply turns each element into a standard record. This can result in strange data. For example, if you have XML elements then only the raw data of each element will be a record. Or for text elements a record with the text element and a record with the corresponding rest element is created. If this is written with FIO.REC() then one line contains the text record and the next line contains the original delimiter with the optional suppressed trailing whitespace in front of it.

ARGUMENTS

- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of records for preallocation [128]

4.3.70 OBJECT *TXT*

SYNOPSIS

```
HELP:   Format elements as text data with delimiters
PATH:   XCNV.OUTPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: TXT(METHOD=HOST/BIN/REC/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM/ORIGINAL, ↩
    CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/ ↩
    UCS4LE/UTF32LE,RPLTAB/RPLHTB=num,RPLVTB=num,RPLCTR=SPACE/SUBSTITUTE/DELETE,SUPTWS,BUFSIZ ↩
    =num,INICNT=num)
```

DESCRIPTION The object "format text" formats text elements into text records with a certain delimiter, no delimiter (mainly useful for record-oriented systems) or produces the original data (concatenate the text element with the corresponding rest element).

To add the delimiter in the correct encoding the charset must be known and can be defined. But normally the charset is already known through the read operation and it is not required to specify this value.

You can replace horizontal and vertical tabs by spaces and new records so that the data looks like it was printed. Additionally, all remaining control characters can be suppressed or replaced by spaces or the substitution character. This features are useful to convert text data containing tabs and other control characters for host data sets.

The method ORIGINAL uses the original delimiter that is present in the input data. If the option suppress trailing whitespace is used, then all whitespace characters in front of the original delimiter are removed.

Text formatting supports a lot of powerful features which can be accessed with the parameters below.

ARGUMENTS

- NUMBER:METHOD=HOST/BIN/REC/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM/ORIGINAL -Method for text formatting [DEFAULT]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - REC -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)

  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
  - ORIGINAL -Adds the original data at the end of a line (only for band FMT)
- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [auto]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:RPLTAB/RPLHTB=num -Replace horizontal tabulators by spaces using this tab width [0 -no replacement]
- NUMBER:RPLVTB=num -Replace vertical tabulators by new lines using this tab width [0 = no replacement]
- NUMBER:RPLCTR=SPACE/SUBSTITUTE/DELETE -Replace remaining control characters [NONE]
  - SPACE -Replace control characters with whitespace character (0x20/0x40)
  - SUBSTITUTE -Replace control characters with substitution character (0x1A/0x3F)
  - DELETE -Remove control characters
- SWITCH:SUPTWS -Suppress trailing whitespaces [FALSE]
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of records for preallocation [128]

4.3.71 OBJECT *XML*

SYNOPSIS

```
HELP:   Format elements (tags, attributes, data, ...) as XML data stream
PATH:   XCNV.OUTPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: XML(METHOD.{})
```

DESCRIPTION The object "format XML" creates a valid XML document from FLAM elements, resulting in UTF-8 data block with character data.

The output XML can be formatted using 4 different methods:

- STANDARD: The XML document is reconstructed exactly as described by the list of FLAM elements (i.e. as close to the original XML document as possible).
- PRETTYPRINT: Creates an XML document with indentation suitable for easy human readability.
- MINIMIZED: Creates a minimized XML document. Line breaks and whitespace are removed unless they are part of actual character data.
- DUMP: Can be used to dump out any arbitrary (even non-XML) FLAM elements and its attributes into an XML document. Can be used to inspect

The XML output is written in blocks and encoded in UTF-8 with line feed (0xA) as line delimiter unless a CCSID is supplied, in which case character conversion is performed.

The PRETTYPRINT and MINIMIZED methods have two parameters that allow control of how to deal with whitespace in input data. In order to decide whether whitespace in the input may be dropped or not, it needs to be cached until the next non-whitespace characters occur. By default, up to 4096 consecutive whitespace characters can be cached. You can increase this number if your documents may contain more consecutive whitespace. You can also specify what to do if the cache runs full. The cached whitespace can either be written to output, possibly resulting in less nicely formatted XML, or processing can be aborted with an error.

An XML document created by the DUMP method has a root tag `<flam>`. It contains one `<configuration>` and one or more `<elementList>` blocks. The `<configuration>` section reflects the options used to create the FLAM elements dump. The `<elementList>` contains one `<element>` tag for each FLAM element in the input data in the order of input. It may have up to three optional attributes (*type*, *attr*, *hash*). The element's data is written to the tag's body. The data formats of the data as well as of the attributes *attr* and *hash* are the ones defined in the `<configuration>` section. Here is a simple example of two FLAM elements dumped to XML with data in hexadecimal format:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<flam>
  <configuration>
    <dumpFormat>
      <data>HEX</data>
      <attr>HEX</attr>
      <hash>HEX</hash>
    </dumpFormat>
  </configuration>
  <elementList>
    <element type="2">01234567890</element>
    <element type="18">0A</element>
  </elementList>
</flam>
```

XML formatting supports a lot of powerful features which can be accessed with the parameters below.

4.3.72 OVERLAY *METHOD*

SYNOPSIS

```
HELP:   Method for XML formatting [STANDARD]
PATH:   XCNV.OUTPUT.SAV.FILE.FMT.XML
TYPE:   OVERLAY
SYNTAX: METHOD.{STANDARD()/PRETTYPRINT()/MINIMIZED()/DUMP()}
```

DESCRIPTION The XML formatting method defines how an XML document will be created and written from FLAM elements containing XML data. It is also possible to dump non-XML elements into an XML document in raw format.

The supported methods are listed below.

The standard method is the default for *write.binary()* and *write.xml()*. Minimized is the default for *write.char()*. Pretty printing is the default for *write.record()* and *write.flam()*.

4.3.73 OBJECT *STANDARD*

SYNOPSIS

```
HELP:   Reproduce all XML elements as is
PATH:   XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD
TYPE:   OBJECT
SYNTAX: STANDARD(NOCMNT,BUFSIZ=num,INICNT=num)
```

DESCRIPTION The standard method recreates the original XML document from FLAM XML elements as close as possible. All document structure, indentation, line breaks, etc. are preserved. All line breaks are normalized to line feeds (0xA) according to the XML specification.

ARGUMENTS

- `SWITCH:NOCMNT -Do not write out XML comments [FALSE]`
- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]`
- `NUMBER:INICNT=num -Initial amount of records for preallocation [128]`

4.3.74 OBJECT *PRETTYPRINT*

SYNOPSIS

```
HELP:   Indentation of XML tags for human readability
PATH:   XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD
TYPE:   OBJECT
SYNTAX: PRETTYPRINT(MAXWSP=num,WSPERR=WRITE/ABORT,INDSIZ=num,INDCHR=SPACE/TABULATOR,NOCMNT, ↩
    BUFSIZ=num,INICNT=num)
```

DESCRIPTION The pretty printing method creates an XML document that is very suitable to be read by humans. All XML elements start on its own line and are properly indented according to its hierarchical position, unless they are surrounded by text data. Elements containing actual text (i.e. they do not only contain whitespace and newlines) have all their whitespace and newline characters preserved.

Example: If the original XML document is the following one:

```
<?xml version="1.0"?>
     <!DOCTYPE root [
  <!ELEMENT root (someelement)>
          ]>
  <root>
                <somelement   attribute1="value1" attribute2="value2">
```

```
    some text
   </somelement>
        <somelement>
    <!-- comment -->
           text in non-root element    <leaf>this is a leaf</leaf></somelement></root>
```

then pretty printing will result in the following document:

```
<?xml version="1.0"?>
<!DOCTYPE root [
  <!ELEMENT root (someelement)>
]>
<root>
  <somelement attribute1="value1" attribute2="value2">
    some text
   </somelement>
  <somelement>
    <!-- comment -->
           text in non-root element    <leaf>this is a leaf</leaf>
  </somelement>
</root>
```

ARGUMENTS

- NUMBER:MAXWSP=num -Size of internal whitespace cache (only used when method is minimize or pretty printing) [4096]
- NUMBER:WSPERR=WRITE/ABORT -Action when the whitespace cache overflows [WRITEOUT]
  - WRITE -Write out cached whitespace on full buffer, causing less nicely formatted output
  - ABORT -Abort XML writing when the whitespace buffer is full, output formatting has priority
- NUMBER:INDSIZ=num -Number of indentation characters per indentation level [2 (SPACE) or 1 (TAB)]
- NUMBER:INDCHR=SPACE/TABULATOR -Indentation character used to indent XML tags [SPACE]
  - SPACE -Use the space character as indentation character
  - TABULATOR -Use the tabulator character as indentation character
- SWITCH:NOCMNT -Do not write out XML comments [FALSE]
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of elements for preallocation [128]

4.3.75 OBJECT *MINIMIZED*

SYNOPSIS

```
HELP:   Discards comments, insignificant whitespace and whitespace-only sections between ↩
    tags
PATH:   XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD
TYPE:   OBJECT
SYNTAX: MINIMIZED(MAXWSP=num,WSPERR=WRITE/ABORT,BUFSIZ=num,INICNT=num)
```

DESCRIPTION The minimized method creates an XML document that has whitespace and linebreaks removed unless they are part of other character data, as well as all comments. This is most suitable if data is primarily contained in the child nodes of the XML document, the document is read by a machine and if storage/bandwidth is an issue.

Example: If the original XML document is the following one:

```
<?xml version="1.0"?>
  <!DOCTYPE root [
    <!ELEMENT root (someelement)>
  ]>
<root>
  <somelement   attribute1="value1" attribute2="value2">
    some text
  </somelement>
  <somelement>
    <!-- comment -->
    text in non-root element
    <leaf>this is a leaf</leaf>
  </somelement>
</root>
```

then minimization will result in the following document:

```
<?xml version="1.0"?><!DOCTYPE root [<!ELEMENT root (someelement)>]><root><somelement  ↩
    attribute1="value1" attribute2="value2">
    some text
  </somelement><somelement>

    text in non-root element
    <leaf>this is a leaf</leaf></somelement></root>
```

ARGUMENTS

- NUMBER:MAXWSP=num -Size of internal whitespace cache (only used when method is minimize or pretty printing) [4096]
- NUMBER:WSPERR=WRITE/ABORT -Action when the whitespace cache overflows [WRITEOUT]
  - WRITE -Write out cached whitespace on full buffer, causing less nicely formatted output
  - ABORT -Abort XML writing when the whitespace buffer is full, output formatting has priority
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of elements for preallocation [128]

4.3.76 OBJECT *DUMP*

SYNOPSIS

```
HELP:   Dumps all elements in raw XML format (also useful for non XML elements)
PATH:   XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD
TYPE:   OBJECT
SYNTAX: DUMP(DATAFMT=BINARY/HEXADECIMAL/NUMERIC,ATTRFMT=BINARY/HEXADECIMAL/NUMERIC,HASHFMT= ↩
    BINARY/HEXADECIMAL/NUMERIC,INDSIZ=num,INDCHR=SPACE/TABULATOR,BUFSIZ=num,INICNT=num)
```

DESCRIPTION The dump method takes lists of arbitrary FLAM elements and dumps them into XML format. For each element list, an <elementList>...<elementList> block is created, containing an arbitrary number of <element>...</element> tags. A FLAM element contains element data, an element type, attributes and a hash, which are represented as

body, `type`, `attr` and `hash` attributes, respectively. The attributes `attr` and `hash` are optional and only written if they contain any data.

The output format of data, attributes and hashes can be configured independantly. Valid formats are binary (may result in a not wellformed XML document), hexadecimal or numeric (bytes printed as decimal numbers separated by spaces). The output format configuration is saved as part of the XML document.

The resulting XML document has the following form:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<flam>
  <configuration>
    <dumpFormat>
      <data>...</data>
      <attr>...</attr>
      <hash>...</hash>
    </dumpFormat>
  </configuration>
  <elementList>
    <element type="..." attr="..." hash="...">...</element>
    <element type="..." attr="..." hash="...">...</element>
    ...
  </elementList>
  <elementList>
    ...
  </elementList>
  ...
</flam>
```

ARGUMENTS

- `NUMBER:DATAFMT=BINARY/HEXADECIMAL/NUMERIC -Format in which element data is dumped [HEX]`
  - `BINARY -Dump in binary format`
  - `HEXADECIMAL -Dump in hexadecimal format`
  - `NUMERIC -Dump in numeric (decimal) format`
- `NUMBER:ATTRFMT=BINARY/HEXADECIMAL/NUMERIC -Format in which attribute data is dumped [HEX]`
  - `BINARY -Dump in binary format`
  - `HEXADECIMAL -Dump in hexadecimal format`
  - `NUMERIC -Dump in numeric (decimal) format`
- `NUMBER:HASHFMT=BINARY/HEXADECIMAL/NUMERIC -Format in which hash data is dumped [HEX]`
  - `BINARY -Dump in binary format`
  - `HEXADECIMAL -Dump in hexadecimal format`
  - `NUMERIC -Dump in numeric (decimal) format`
- `NUMBER:INDSIZ=num -Number of indentation characters per indentation level [2 (SPACE) or 1 (TAB)]`
- `NUMBER:INDCHR=SPACE/TABULATOR -Indentation character used to indent XML tags [SPACE]`
  - `SPACE -Use the space character as indentation character`
  - `TABULATOR -Use the tabulator character as indentation character`
- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]`
- `NUMBER:INICNT=num -Initial amount of elements for preallocation [128]`

4.3.77 OVERLAY *CNV*

SYNOPSIS

```
HELP:   Conversion [NONE]
PATH:   XCNV.OUTPUT.SAV.FILE
TYPE:   OVERLAY
SYNTAX: CNV[{BLK()/REC()/GZP()/BZ2()/LXZ()/CHR()/BAS()/HSH()}...]
```

DESCRIPTION With the overlay CNV you can choose the different kind of conversions supported at write operations.

4.3.78 OBJECT *BLK*

SYNOPSIS

```
HELP:   Convert a record list to one block (binary or text) of data
PATH:   XCNV.OUTPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: BLK(BUFSIZ=num,METHOD=HOST/BIN/REC/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM ↩
    /L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX,CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/ ↩
    EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE,SUPTWS,RPLTAB/RPLHTB= ↩
    num,RPLVTB=num,RPLCTR=SPACE/SUBSTITUTE/DELETE)
```

DESCRIPTION The block converter can be used to convert records into data blocks with delimiters. On output, it can also add different length formats in front of the records. This conversion component uses the text, block and binary formatting components and provides all their features.

ARGUMENTS

- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:METHOD=HOST/BIN/REC/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM/L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX -Method for block/text formatting [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - REC -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
  - L4I -Adds 4 byte length fields:Little endian integer, length inclusive
  - L4X -Adds 4 byte length fields:Little endian integer, length exclusive (ZIP)
  - B4I -Adds 4 byte length fields:Big endian integer, length inclusive
  - B4X -Adds 4 byte length fields:Big endian integer, length exclusive (USS)
  - S4I -Adds 4 byte length fields:System endian integer, length inclusive
  - S4X -Adds 4 byte length fields:System endian integer, length exclusive (VAR)

  - HLI -Adds 4 byte length fields:Little endian short (LL00), length inclusive
  - HLX -Adds 4 byte length fields:Little endian short (LL00), length exclusive
  - HBI -Adds 4 byte length fields:Big endian short (LL00), length inclusive (MVS)
  - HBX -Adds 4 byte length fields:Big endian short (LL00), length exclusive
  - HSI -Adds 4 byte length fields:System endian short (LL00), length inclusive
  - HSX -Adds 4 byte length fields:System endian short (LL00), length exclusive
- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- SWITCH:SUPTWS -Suppress trailing whitespace [FALSE]
- NUMBER:RPLTAB/RPLHTB=num -Replace horizontal tabulators by spaces using this tab width [0 =no replacement]
- NUMBER:RPLVTB=num -Replace vertical tabulators by new lines using this tab width [0 = no replacement]
- NUMBER:RPLCTR=SPACE/SUBSTITUTE/DELETE -Replace remaining control characters [NONE]
  - SPACE -Replace control characters with whitespace character (0x20/0x40)
  - SUBSTITUTE -Replace control characters with substitution character (0x1A/0x3F)
  - DELETE -Remove control characters

4.3.79 OBJECT *REC*

SYNOPSIS

```
HELP:   Convert a block (binary or text) of data into a list of records
PATH:   XCNV.OUTPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: REC(METHOD=BIN/DLM/REC,CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/ ↩
    UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE,RPLFFD=num,RPLTAB/RPLHTB=num,RPLVTB=num, ↩
    RPLCTR=SPACE/SUBSTITUTE/DELETE,SUPTWS,NELDLM,PADCHR=num,RECLEN=num,BUFSIZ=num,INICNT=num ↩
    )
```

DESCRIPTION The record converter can be used to convert data blocks with delimiters or on input with different record length formats into data lists with records. This conversion component uses the text and record formatting component and provides all its features.

Auto detection of 4 byte record length fields is attempted if no method is provided. If the data block contains no record length fields and the switch to skip text formatting is not enabled, then all capabilities of the text formatting component are used to form records.

The SKPTXT flag causes input blocks to be only converted to records if record length fields are detected. This can be useful, for example, for reading ZIP archives containing record-oriented files. PKZIP stores record-oriented files with a 4 byte length field (length of the field itself not included) in front of the record data in little endian (L4X). Another example files stored with FILE-DATA=RECORD (B4X) in USS. When reading such a file, the length fields must be interpreted to re-build the corresponding records. This conversion can be added after any I/O or other conversion step.

ARGUMENTS

- NUMBER:METHOD=BIN/DLM/REC -Method for record/text formatting [DEFAULT]
  - BIN -Wrap data block in records of record length
  - DLM -Parse data block for text delimiter (DEFAULT)
  - REC -Parse data block based on record length fields
- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assumin g n lines per page [0 =no replacement]
- NUMBER:RPLTAB/RPLHTB=num -Replace horizontal tabulators by spaces using this tab wid th [0 =no replacement]
- NUMBER:RPLVTB=num -Replace vertical tabulators by new lines using this tab width [0 = no replacement]
- NUMBER:RPLCTR=SPACE/SUBSTITUTE/DELETE -Replace remaining control characters [NONE]
  - SPACE -Replace control characters with whitespace character (0x20/0x40)
  - SUBSTITUTE -Replace control characters with substitution character (0x1A/0x3F)
  - DELETE -Remove control characters
- SWITCH:SUPTWS -Suppress trailing whitespaces [FALSE]

- SWITCH:NELDLM -Activate NEL (0x85) as delimiter for ASCII character sets [FALSE]
- NUMBER:PADCHR=num -Padding character [0x00]
- NUMBER:RECLEN=num -Length used to cut the data block in records [512]
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of records for preallocation [128]

4.3.80 OBJECT *GZP*

SYNOPSIS

```
HELP:   Write GZIP/RCF1952 compressed data
PATH:   XCNV.OUTPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: GZP(BUFSIZ=num,CMPLEV=num/COPY/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/ ↩
    BEST/AUTO,TXTDAT,MODTIM=num,OPRSYS=FAT/AMIGA/VMS/UNIX/VMCMS/ATARI/HPFS/MAC/ZOS/CPM/ ↩
    TOPS20/NTFS/QDOS/ACORN/UNKNOWN,EXTRAD='bin',MBRNAM='str',COMENT='str',HDRCHK)
```

DESCRIPTION The GZIP converter can be used to compress or decompress data streams.

The result of a compression is a binary block. The result of a decompression can be text, XML, a base encoding or binary data. Record boundaries are lost. Please use delimiters if records must be exchanged.

The GZIP decompression can read all supported formats of zlib and writes RFC1952/51/50-compliant data streams. If a IBM ZEDC acceleration card is available and can be used, the ZEDC is used to save CPU time. In this case, however, the compression does not conform to RFC1951. The compressed data is encoded based on RFC1950 and the file format conforms to RCF1952.

For a checksummed GZIP header, an additional switch must be activated, because this feature is not supported by all GZIP implementations. The GZIP header is used to store meta information about the data, which can be used if FLAM is also used for decompression. This can be very useful to clearly define the character set, CCSID and other information about the data.

If no compression level is set and the data is marked as being without redundancy, then level 0 (copy) is used. Otherwise, the default level 6 is used.

Decompression of concatenated GZIP files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
   filename/:memberindex
   filename/#memberindex

   test.gz/#3
   test.gz/:2
```

GZIP data streams are written to ZIP archives as compression mode 8. The GZIP header and trailer are replaced by a ZIP local file header. If a deflated file (compression mode 8) is read from a ZIP archive, it is converted to a GZIP data stream which can be inflated with this component.

ARGUMENTS

- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:CMPLEV=num/COPY/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level [AUTO]
  - COPY -No compression =copy of data (0)
  - FAST -Fastest compression (level 1)
  - LEV1 -Compression level 1
  - LEV2 -Compression level 2

  - LEV3 -Compression level 3
  - LEV4 -Compression level 4
  - LEV5 -Compression level 5
  - LEV6 -Compression level 6
  - LEV7 -Compression level 7
  - LEV8 -Compression level 8
  - LEV9 -Compression level 9
  - BEST -Best compression (level 9)
  - AUTO -Automatic compression (level 6)
- SWITCH:TXTDAT -Mark content as text data in header [from origin]
- NUMBER:MODTIM=num -Modification time in header (seconds since 1970) [from origin]
- NUMBER:OPRSYS=FAT/AMIGA/VMS/UNIX/VMCMS/ATARI/HPFS/MAC/ZOS/CPM/TOPS20/NTFS/QDOS/ACORN/UNKNOWN -Operation/file system in header [from origin]
  - FAT -GZIP operating/file system FAT
  - AMIGA -GZIP operating/file system AMIGA
  - VMS -GZIP operating/file system VMS
  - UNIX -GZIP operating/file system UNIX
  - VMCMS -GZIP operating/file system VMCMS
  - ATARI -GZIP operating/file system ATARI
  - HPFS -GZIP operating/file system HPFS
  - MAC -GZIP operating/file system MAC
  - ZOS -GZIP operating/file system ZOS
  - CPM -GZIP operating/file system CPM
  - TOPS20 -GZIP operating/file system HPFS
  - NTFS -GZIP operating/file system NTFS
  - QDOS -GZIP operating/file system QDOS
  - ACORN -GZIP operating/file system ACORN
  - UNKNOWN -GZIP operating/file system UNKOWN
- STRING:EXTRAD='bin' -Extra data for header
- STRING:MBRNAM='str' -Rename filename for header [from origin]
- STRING:COMENT='str' -Comment for header
- SWITCH:HDRCHK -Use checksum for header [FALSE]

4.3.81 OBJECT *BZ2*

SYNOPSIS

```
HELP:   Write BZIP2 compressed data
PATH:   XCNV.OUTPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: BZ2(BUFSIZ=num,CMPLEV=num/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/ ↩
    AUTO,WRKFAC=num)
```

DESCRIPTION The BZIP2 converter can be used to compress or decompress data streams compatible with the BZIP2 utility on UNIX systems.

The result of a compression is a binary block. The result of a decompression can be text, XML, a base encoding or binary data. Record boundaries are lost. Please use delimiters if records must be exchanged.

There is no header to store meta information about the data, so all metadata is lost.

If no compression level is defined, the level used depends on the block size. A block size larger than 900KB results in the best compression level 9, greater than 800KB and smaller than 900KB in level 8, and so on. For blocks smaller than 100KB level 1 is used.

Decompression of concatenated BZIP2 files is also possible. All concatenated files will be decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
   filename/:memberindex
   filename/#memberindex

   test.bz/#3
   test.bz/:2
```

BZIP2 data is written to ZIP archives as compression mode 12. The BZIP2 header and trailer are replaced by a ZIP local file header. If a BZIP2 file (compression mode 12) is read from a ZIP archive, it is converted to a BZIP2 data stream and can be decompressed with this component.

ARGUMENTS

- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:CMPLEV=num/FAST/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level, defines block size for compression strength [AUTO]
  - FAST -Fastest compression (Block size =100kb)
  - LEV1 -Compression level 1 (Block size =100kb)
  - LEV2 -Compression level 2 (Block size =200kb)
  - LEV3 -Compression level 3 (Block size =300kb)
  - LEV4 -Compression level 4 (Block size =400kb)
  - LEV5 -Compression level 5 (Block size =500kb)
  - LEV6 -Compression level 6 (Block size =600kb)
  - LEV7 -Compression level 7 (Block size =700kb)
  - LEV8 -Compression level 8 (Block size =800kb)
  - LEV9 -Compression level 9 (Block size =900kb)
  - BEST -Best compression (Block size =900kb)
  - AUTO -Automatic compression (depending on the real block size)
- NUMBER:WRKFAC=num -Work factor from 0 to 250 for highly repetitive data handling [0==30]

4.3.82 OBJECT *LXZ*

SYNOPSIS

```
HELP:   Write XZ (LZMA) compressed data
PATH:   XCNV.OUTPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: LXZ(BUFSIZ=num,CMPLEV=num/FAST/LEV0/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/ ↩
    BEST/AUTO)
```

DESCRIPTION The LZMA/XZ converter can be used to compress or decompress data streams compatible with the XZ utility on UNIX systems.

The result of a compression is a binary block. The result of a decompression can be text, XML, a base encoding or binary data. Record boundaries are lost. Please use delimiters if records must be exchanged.

There is no header to store meta information about the data, so all metadata is lost.

If no compression level is defined and the data is marked as being without redundancy, then level 0 (copy) is used. In all other cases the default level 6 is used.

Decompression of concatenated XZ files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
filename/#memberindex

test.xz/#3
test.xz/:2
```

XZ data streams are written to ZIP archives as compression mode 14 (LZMA). The XZ header and trailer are replaced by a ZIP local file header. If an LZMA file (compression mode 14) is read from a ZIP archive, it is converted to an XZ data stream which can be decompressed with this component.

ARGUMENTS

- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:CMPLEV=num/FAST/LEV0/LEV1/LEV2/LEV3/LEV4/LEV5/LEV6/LEV7/LEV8/LEV9/BEST/AUTO -Compression level [AUTO]
  - FAST -Fastest compression (level 0)
  - LEV0 -Compression level 0
  - LEV1 -Compression level 1
  - LEV2 -Compression level 2
  - LEV3 -Compression level 3
  - LEV4 -Compression level 4
  - LEV5 -Compression level 5
  - LEV6 -Compression level 6
  - LEV7 -Compression level 7
  - LEV8 -Compression level 8
  - LEV9 -Compression level 9
  - BEST -Best compression (level 9)
  - AUTO -Automatic compression (level 6)

4.3.83 OBJECT *CHR*

SYNOPSIS

```
HELP:   Convert character sets (ASCII, EBCDIC, UNICODE)
PATH:   XCNV.OUTPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: CHR(BUFSIZ=num,RECCNT=num,METHOD=AUTO/BLOCK/RECORD/KEEP/SKIP,CASMOD=UPPER/LOWER/ ↩
    FOLD/SUPPER/SLOWER/USRTAB,FROM='str'/DEFAULT/ASCII/EBCDIC,MODE=STOP/IGNORE/SUBSTITUTE/ ↩
    IDENTITY,TO='str'/DEFAULT/ASCII/EBCDIC,WRTBOM/BOM,HDLBOM,ENL2LF,ELF2NL,SUBCHR[num/SYSTEM ↩
    ...],SYSTAB=ICONV,USRTAB='str'/NPAS/SEPA/DELA/DLAX,REPFIL='str'/STDOUT/STDERR,SKPBIN, ↩
    SKPEQU)
```

DESCRIPTION This component can be used to convert a text stream to a different character set after reading or before writing. Character conversion requires a source CCSID (FROM) and a target CCSID (TO). Using the keyword MODE, you can define the behavior when invalid or incomplete characters are encountered in the input data. It is possible to stop at, ignore or substitute unsupported characters. For the latter case, you can define a substitution character list (SUBCHR) and/or a user substitution table (USRTAB) and/or use one of the pre-defined system substitution tables (SYSTAB) for transliteration. When using a system table, you can override some of the pre-defined code points using a user table. Each character must be defined as UNICODE point in hexadecimal notation.

Each ignore or substitution process, including transliteration, can be recorded in a report file for review. Moreover, you can convert to upper, lower, special case or use case folding. The component also provides powerful support for byte order marks (BOM) including:

- Byte order change handling at read (for example UTF-16LE in UTF-16BE)
- Byte order marks per stream (if block oriented) or record at write
- Charset detection based on byte order marks

You can differentiate between block- and record-oriented conversion by the keyword METHOD.

- record oriented conversion - throws an error if an incomplete byte sequence occurs at the end of a record
- block oriented conversion - no check for record boundaries, regards all data as one block

If the input text is record-oriented then record orientation will be used by default, and you must explicitly enable block-oriented conversion if you do not intend to check the data at each end of a record. If you choose block orientation, the record lengths table will be lost. Additionally, you can enforce to skip incomplete character sequences at the end of a record to keep the length information.

In case of an error, all position and counter values printed to the error trace contain the current indexes. All indexes start at zero. The first block or record is unit 0.

It supports replacement of EBCDIC new line (0x15) characters by line feed (0x25) at character conversion takes place. When converting to UTF-8/16/32, this will result in the new line characters to be converted to line feeds (0x0A) instead of UTF-8 next line characters (0xC285).

In read mode, the component supports auto-detection of charsets. This can be activated by using the keyword DEFAULT as CCSID. Auto-detection will analyze the first block of input data to determine if the input uses ASCII, EBCDIC or a UTF encoding. The statistic-based computation cannot determine the actual ASCII or EBCDIC character set. Instead, the final CCSID is derived from the language identifier of the environment variable LANG.

Character conversion can skipped or enforced if the CCSIDs are equal or or binary data is detected.

To get a list of supported CCSIDs, use the command below:

```
   flcl INFO GET.CCSID
```

For more information, please have a look at the man pages for each parameter.

ARGUMENTS

- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]`
- `NUMBER:RECCNT=num -Initial amount of records for preallocation [128]`

4.3.84 PARAMETER *METHOD*

SYNOPSIS

```
HELP:   Enforces record or block-oriented conversion [AUTO]
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: METHOD=AUTO/BLOCK/RECORD/KEEP/SKIP
```

DESCRIPTION The method of character conversion defines how the input data fields are handled. A data field can be one block or a number of records dependent from how the data is read and converted before character conversion is done. Is the data field a list of records then the record oriented method shall be used. Is the data field a single block the block oriented conversion shall be used. This means that the METHOD must only be set to enforce record or block orientation against the usual way of processing the data.

- Block Orientation - This means all records are handled as one block and the record length information will go lost. If incomplete characters occur at the end of this block it will be remembered as rest and copied to the next block. Using this method there will be no interruption of the data stream.
- Record Orientation - This method ensures that each record contains only complete character sequences. An incomplete character sequence will only be accepted at the end of the last record and handled also as rest for the record in the next data field. To enforce record orientation processing of one block of data there will be no difference to block orientation conversion, besides for processing the last block.

Thus it makes only sense to switch for a record organized data field to block orientation conversion. But this is very critical, as the length information will go lost. This means that there will be no chance to retrieve the record information, besides the record contains some delimiters which can be used to build a new list of record lengths. So we recommend to use the METHOD switch only if you know what you are doing. In some cases analysis of the error trace might result in the correct method that can be used to convert the text.

If an incomplete byte sequence progresses in the next record, then it can be helpful to enforce block oriented conversion, because the record length conversion can be invalid. On the other side, if the next record starts with a new valid byte sequence, then a skip of the damaged byte sequence in the last record can be helpful.

If SKIP is selected then each incomplete character at the end of a record is recorded in the report file with a special code point 0xFFFFFF including the really skipped byte sequence. If this problem happens in the last block you can enforce record oriented conversion with SKIP to convert the data and the incomplete rest of the last block. They are written to the report file. Normally the corresponding MODE is used for the final block to prevent a rest.

Please don't mix up SKIP as METHOD for the rest of a record contained in a data field and the IGNORE as MODE for the content of a record as unit of work.

The default method depends on the data. If the data is marked as block-oriented or the record type is text with delimiters, then the method BLOCK is used. In all other cases, record orientation with the method KEEP is used.

SELECTIONS

- `AUTO -Automatic orientation (block or record with keep)`
- `BLOCK -Block-oriented conversion (all data is one chunk)`
- `RECORD -Record-oriented conversion (check record boundaries)`
- `KEEP -Keep (Move a rest from the end to begin of next record)`
- `SKIP -Skip a rest at the end of a record (not recommended)`

4.3.85 PARAMETER *CASMOD*

SYNOPSIS

```
HELP:   Define upper, lower or special case conversion [NONE]
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: CASMOD=UPPER/LOWER/FOLD/SUPPER/SLOWER/USRTAB
```

DESCRIPTION With the case modifier, you can perform conversion to upper or lower case as well as apply case folding and some special case conversions.

Case folding converts all characters to lower case but is more compatible with language specific rules for doing so. For example, this may be useful for case-insensitive string comparison if the strings contain non-Latin characters (code point < 128).

This mapping is done for each code point in front of substitution or transliteration. The respective case is accomplished on the basis of a separate Unicode conversion table in memory. This conversion can be manipulated if user mapping is activated and mapping definitions (⋆) are defined through the provided user table.

The options UPPER and LOWER convert characters without expansion, i.e. returned strings will never grow in length. Note, however, that case folding and the special options (convert e.g. ß to SS) can result in an expansion of data.

All these conversions are pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `UPPER -Change character to upper case`
- `LOWER -Change character to lower case`
- `FOLD -Change character to folded case (for compare)`
- `SUPPER -Change character to special upper case`
- `SLOWER -Change character to special lower case`
- `USRTAB -Change character according to user table`

4.3.86 PARAMETER *FROM*

SYNOPSIS

```
HELP:   Conversion from this CCSID [DEFAULT]
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   STRING
SYNTAX: FROM='str'/DEFAULT/ASCII/EBCDIC
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT,

ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (UTF-8)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`

4.3.87 PARAMETER *MODE*

SYNOPSIS

```
HELP:   Define handling of non-convertible characters
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: MODE=STOP/IGNORE/SUBSTITUTE/IDENTITY
```

DESCRIPTION With the MODE selection you can define how an invalid or incomplete character is handled inside of a unit by this component. All possible modes are listed at the end of this chapter.

This MODE is valid for the content of each unit (block or record). At the end of a unit there can be an incomplete character. Such a rest will be handled dependent on the METHOD you choose. The rest of the last unit of a data field will be concatenated to the first unit of the next data field. For all other records in a data field the handling of such a rest will be defined by the respective METHOD.

The SKIP case for METHOD is only relevant for incomplete byte sequences at the end of a unit. The ignore and substitution mode is only effective for all characters in the middle of the unit, besides an incomplete byte sequence occurs in the last unit. Please don't mix up the IGNORE case for MODE with the SKIP case of METHOD.

If a system or user substitution table is used the stop, ignore or substitute are only relevant for characters not managed by the CCSIDs and the substitution tables. This means that the USRTAB and SYSTAB are added to the capabilities of character conversion and only for the rest the chosen MODE is used.

4.3.87.1 CONSTANT *STOP*

SYNOPSIS

```
HELP:   Stop at the first non-convertible character
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR.MODE
TYPE:   NUMBER
SYNTAX: STOP
```

DESCRIPTION Stops conversion at the first incomplete or invalid character which can not be converted. An error message is given with the position of abortion and the value in the input stream. An entry will be (STP()) written to the report file to log this event. To be compatible with other conversion tools, this is the default case, but we recommend to use IGNORE or SUBSTITUTE to prevent any abortion.

indexterm:[Constant STOP}

4.3.87.2 CONSTANT *IGNORE*

SYNOPSIS

```
HELP:   Ignore non-convertible characters
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR.MODE
TYPE:   NUMBER
SYNTAX: IGNORE
```

DESCRIPTION Ignores incomplete or invalid characters and writes a report entry for each character. IGNORE is a substitution by no corresponding character that means no substitution table is required.

indexterm:[Constant IGNORE}

4.3.87.3 CONSTANT *SUBSTITUTE*

SYNOPSIS

```
HELP:   Substitute non-convertible characters
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR.MODE
TYPE:   NUMBER
SYNTAX: SUBSTITUTE
```

DESCRIPTION Substitutes incomplete or invalid character and writes a report entry for each character. For SUBSTITUTE you can define a substitution character list (SUBCHR). If no SUBCHR is provided then this option substitutes invalid or incomplete characters with code point 0x00001A. If a SUBCHR is defined all invalid or incomplete characters will be substituted by the defined list of code points that can be up to 8. If you specify a SUBCHR the substitution mode will be enabled.

indexterm:[Constant SUBSTITUTE}

4.3.87.4 CONSTANT *IDENTITY*

SYNOPSIS

```
HELP:   Copy non-convertible characters (supported only for single byte conversions)
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR.MODE
TYPE:   NUMBER
SYNTAX: IDENTITY
```

DESCRIPTION Copies invalid characters from input to output and writes a report entry for each character. Resulting output could be not printable.

indexterm:[Constant IDENTITY}

4.3.88 PARAMETER *TO*

SYNOPSIS

```
HELP:   Conversion to this CCSID [DEFAULT]
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   STRING
SYNTAX: TO='str'/DEFAULT/ASCII/EBCDIC
```

DESCRIPTION The TO code defines the encoding string or CCSID for the output text stream. The default value of this CCSID for read operations is UTF-8. For write operations, the default value is derived from the environment variable LANG. If LANG is not set, the default ASCII or EBCDIC system-codepage is used, depending on the platform.

To get a list of supported encoding strings, please use the command below:

```
   flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs please use:

```
   flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
   CCSID='37'    will be equal to CCSID='IBM037'
   CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit. Optionally, you can use one of the keywords below to use the system-specific default ASCII or EBCDIC CCSID.

SELECTIONS

- `DEFAULT -Use default CCSID (system-specific)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`

4.3.89 PARAMETER *WRTBOM*

SYNOPSIS

```
HELP:   Write byte order mark in front of the output text
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: WRTBOM/BOM
```

DESCRIPTION If the write byte order mark keyword is defined, then the corresponding BOM sign is written at the start of the output stream (block oriented) or at the beginning of each record. If this BOM keyword is not used, then the BOM sign will not be written (this is valid independent of the chosen encoding/CCSID). If a valid BOM sign is located in the middle of the input stream, it is removed from the data. If you define the BOM keyword and the target charset contains only single bytes or no valid BOM sign, then no byte order mark will be written to the output file.

If an invalid BOM sign is recognized, then it will be ignored and causes no entry in the report file. If the byte order changes in the middle of text and the BOM handling switch is not activated, the conversion will stop and an error message is thrown.

By default, the write BOM keyword is set to OFF.

4.3.90 PARAMETER *HDLBOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: HDLBOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.3.91 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25)
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.3.92 PARAMETER *ELF2NL*

SYNOPSIS

```
HELP:   Replace EBCDIC line feed (0x25) by new line (0x15)
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: ELF2NL
```

DESCRIPTION The replacement of EBCDIC line feed control characters (0x25) by new line characters (0x15) at conversion to EBCDIC can be activated on non EBCDIC (ASCII) systems or deactivated on EBCDIC systems by enabling this switch. This is for example required to prepare text files on open platforms for USS on z/OS. The usual delimiter for text in ASCII and UTF is 0x0A and will normally be translated to 0x25 in EBCDIC. On z/OS(USS), however, 0x15 (NL) is expected.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

To ensure a correctly delimited text record, only a line feed and not a carriage return (0x0D) line feed (0x0A) is used. If text formatting is used with method=NL, this will result in a single LF on distributed systems and the internal conversion will be activated by default to produce a valid text file for z/OS under USS on Windows, Unix and other open platforms.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.3.93 PARAMETER *SUBCHR*

SYNOPSIS

```
HELP:   Default substitution character list [0x1A] (maximal 8 code points)
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: SUBCHR[num/SYSTEM...]
```

DESCRIPTION As substitution character (SUBCHR) you can define a list with up to 8 UNICODE points which are used for the substitution of invalid or incomplete characters if the corresponding code point is not defined in the substitution table (see SUBTAB). If you assign to it the keyword SYSTEM the value 0x1A is written as default substitution character.

If you activate substitution and no SUBCHR is defined, then unsupported characters will be replaced by CODEPOINT 0x00001A.

On the other hand SUBCHR will replace every undefined code point in the substitution table, what is the same behavior when using a USRTAB with corresponding SUBCHR values for every possible code point (000000-200000). If SUBCHR but no MODE is defined the substitution mode will be activated.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `SYSTEM –Use system specific substitution char (0x1A for ASCII/UTF, 0x3F for EBCDIC)`

4.3.94 PARAMETER *SYSTAB*

SYNOPSIS

```
HELP:   Define a preloaded system substitution table
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: SYSTAB=ICONV
```

DESCRIPTION With the system table (SYSTAB) you can load a substitution table with a predefined transliteration schema. Entries in the USRTAB can be used to override predefined transliteration entries or add missing transliteration code points. The options listed at the end of this chapter are available.

If a substitution table is used then the MODE option (STOP/IGNORE/SUBSTITUTE) is only relevant for characters that cannot be converted or substituted.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `ICONV –Preload with ICONV//TRANSLIT substitution table`

4.3.95 PARAMETER *USRTAB*

SYNOPSIS

```
HELP:   Name of the file containing the user conversion table
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   STRING
SYNTAX: USRTAB='str'/NPAS/SEPA/DELA/DLAX
```

DESCRIPTION With this parameter, you can choose a predefined subset or standard manipulation for the system tables via keyword (see `flcl help "xcnv.inp .sav.fil.cnv.chr.usrtab"` for available keywords). Alternatively, you can use a file containing a user defined substitution table (USRTAB).

For the user table file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
USRTAB=NPAS                          ; use XOEV predefined subset
USRTAB=DELA                          ; German Latin single byte subset
USRTAB='~.MYUSRTAB(USRTAB1)'         ; PDS on z/OS
USRTAB='<SYSUID>.MYUSRT02'           ; PS dataset on z/OS
USRTAB='~/myusrtab3.txt'             ; Unix path name
USRTAB='<HOME>\usrtab\myusrtab4.txt' ; Windows path name
```

A user defined substitution table has the syntax listed below:

```
usr_tab_file         -> usr_definition_list
usr_definition_list  -> usr_definition usr_definition_list
                     |  EMPTY
usr_definition       -> '(' codepoint_definition ')'
codepoint_definition -> cp_activation
                     |  cp_deactivation
                     |  cp_transliteration
                     |  cp_standard_mapping
                     |  cp_usr_case_mapping
cp_activation        -> ['+'] CODEPOINT  [ '-' CODEPOINT ]
cp_deactivation      ->  '-'  CODEPOINT  [ '-' CODEPOINT ]
cp_transliteration   -> ['+'] CODEPOINT '=' code_point_list
cp_standard_mapping  ->  '*'  CODEPOINT '=' code_point_list
cp_usr_case_mapping  ->  '^'  CODEPOINT '=' code_point_list
code_point_list      -> CODEPOINT separator code_point_list
                     |  CODEPOINT
                     |  EMPTY
separator            -> '/'
                     |  ','
                     |  ';'
```

Note: In the description below "/" is used as code point separator.

The optional signs in front of a code point (CP) have the following meaning:

```
'+' - a valid transliteration (only if CP not in the target set)
'*' - a valid mapping         (this translation is always done)
'^' - a case mapping/folding  (only if case=usrtab is activated)
'-' - an invalid definition   (removes CP for subset definitions)
```

If no sign is used, then a transliteration for a valid code point is defined. Valid code points are accepted, invalid code points result in in an error. Error handling depends on the mode (STOP,IGNORE, SUBSTITUTE). Definition of + code points without code point list, thus without = is equal definition of valid code points. With an asterisk $*$, you can define an enforced mapping for this code point. For example, this can be used to delete this character if no code point list is provided or to translate this character always in another value. This can also be used to convert code points outside of a subset into this subset.

In contrast to the transliteration (+ is only done if a character doesn't exist in the target encoding), the mapping is always done for each character. Invalid (−) code points can be used to deactivate a code point. To activate (+) or deactivate (−) a range of code points you can use the minus mark (−) followed by one sub code point. The range goes from the smaller code point to the bigger one. For example:

```
(-00-7F) deactivate all US-ASCII code points
(+39-30) activate all decimal digits
```

The range operator (−) with the optional plus sign (+) can also be used to activate code points.

To define an own case mapping or folding (required for certain subsets) the caret sign (^) can be used. This mapping is only done if the user table is activated for case mapping (CASE=USRTAB).

To define transliterations/substitutions/mappings an assignment (=) of a code point list is required. The code point list can contain a maximum of 8 code points. If no code point list is specified, then the character is ignored. A transliteration to itself simply activates this code point. The same is valid for mapping definitions. A code point is a hexadecimal number representing a 21 bit UNICODE point.

```
  00000000 to 001FFFFF hex
```

If a code point is not defined in the substitution table (USRTAB and/or SYSTAB), the appropriate substitution character (SUBCHR) is used. If SUBCHR is not set, no substitution is performed. Depending on the MODE, character conversion stops with an error (STOP) or ignores the character (IGNORE).

Text before and after brackets are comments.

Between "(" and "=" or "/" or ")" you can define hex digits until the first non-hex digit. All non-hex digits up to the next separator are interpreted as comment. Leading whitespace is ignored.

```
  REPLACE GERMAN SZ (00DF=0073/0073)                  WITH ss
THIS IS AN EXAMPLE FOR COMMENTS IN CODE POINT LIST:
  REPLACE EURO MARK (20AC= 45#E / 55#U / 52#R / 4F#O) WITH EURO
  REPLACE BOM MARK  (EFFF=)                           WITH NOTHING
```

Leading zeros are allowed. If you don't supply a hex value, then 0x00 is used.

```
  REPLACE GERMAN SZ (00DF=/)                   WITH 0x00 0x00
  REPLACE EURO MARK (20AC=00000045/00055/000052/000004F) WITH EURO
```

ATTENTION: Please don't use parentheses "()" or the operators in your comments.

To describe your own subsets, you can use a user table without a system table. The USRTAB can also be used to overwrite or add transliterations when a system substitution table (for example SYSTAB=ICONV) is used. The transliteration works recursively, that is if one of the substitution code points is not in the target set, this substitution will be used instead and so on.

```
  REPLACE GERMAN OE (0000D6=004F/0045)       WITH O E
  REPLACE GERMAN SZ (0000DF=0073/0073)       WITH s s
  REPLACE EURO MARK (0020AC=00D6/00DF/52/4F) WITH OE SZ R O
```

If you have a EURO sign in your text and convert it to *Latin1*, the resulting byte string will be *D6DF524F*. On the other hand, if you convert it to *ASCII*, the byte string will be *4F457373524F*.

The mapping as described above is always done and is also done recursive including the transliteration result. For example, if you define a mapping `(*30=39)` then the target data won't have any zero in the resulting text.

By replacing other code points recursively, you could easily cause infinite loops. To prevent this, the amount of replacements and the length of one replacement is limited to a maximum of 64. Note that this could still result in a large expansion of data. Therefore, be careful about defining recursive substitutions.

A sample user table for the ICONV system table to change the transliteration of German umlauts to AE, OE or UE is located in the SAMPLE directory under CCUTDEXL. Another sample user table called CCUTNPAS defines the *string.latin* subset (XOEF) which is mainly used for statutory reporting.

The order of definition is the order of processing the definitions. For exampel, if you define a mapping or a transliteration for code point X and later you make this code point invalid, then the code point is invalid. On the other hand, you can first deactivate all code points, then activate your subset and define your transliterations (best fit) or mappings.

For example: To define a non-expansion best fit mapping a user table can be used to delete all combined characters for single byte code pages. For the ICONV system transliteration table, a sample user table can be found under the name CCUTBFM1.

Some other subset user table definitions were added. For example:

- currently valid SEPA characters (<128) for money transfers (CCUTSEPA)
- characters valid in ISO-8859-15, CP1252 and IBM1142 (CCUTDELA)
- characters valid in ISO-8859-15, CP1252, IBM1142 and XOEF (CCUTDLAX)

All user table definitions are pre-calculated once at the beginning of execution to reduce the required CPU time for the conversion. So, the use of a user table increases the cost to open a file but has no effect on CPU utilization when converting the data.

To minimize the effort and to simplify the usage of user table, the most common subsets can be selected by keyword. In this case, a pre-calculated load module (DLL) is used. The subset definition can be for UTF (21 Bit UNICODE) or UCS (16 Bit UNICODE). If it is a UCS subset, then no valid code point is greater than 0xFFFF and you must use UCS CCSIDS to ensure that no code point greater than 0xFFFF is accepted.

If you have a working user table definition for a popular subset or system table manipulation, you can request a keyword for it. Please report this requirement over our issue tracking system

```
http://www.flam.de/en/technology/support/issues/
```

and attach the corresponding user table definition.

SELECTIONS

- `NPAS –UCS subset for statutory reporting (New passport, XOEF, String.Latin)`
- `SEPA –UCS subset of valid SEPA character smaller then 128`
- `DELA –UCS subset for German Latin of IBM1141, ISO8859-15 and CP1252`
- `DLAX –UCS subset for German Latin of IBM1141, ISO8859-15, CP1252 and XOEF`

4.3.96 PARAMETER *REPFIL*

SYNOPSIS

```
HELP:   Name of the report file to log substitutions
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   STRING
SYNTAX: REPFIL='str'/STDOUT/STDERR
```

DESCRIPTION With the report capability, you can log every substitution (stop / ignore / substitute / translit / skip / . . .) to a log file. For the report file, all rules and replacements are valid, which are described under "FILENAME HANDLING" above. For example:

```
REPORT=STDERR                          ; is SYSPRINT on z/OS
REPORT='~.MYREPORT(REPORT1)'           ; PDS on z/OS
REPORT='<SYSUID>.MYREPRT2'             ; PS dataset on z/OS
REPORT='~/myreport3.txt'               ; Unix path name
REPORT='<HOME>\reports\myreport4.txt'  ; Windows path name
```

A substitution report has the syntax listed below:

```
report_file         -> sub_definition_list
sub_definition_list -> sub_definition sub_definition_list
                    |  EMPTY
sub_definition      -> SUB '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    -> XLT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    |  IGN '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  STP '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  IDT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  SKP '(' UNIT ':' OFFSET '+' POSITION ':'
```

```
                          code_point byte_string ')'
                    |  UKW '(' UNIT ':' OFFSET '+' POSITION ':'
                          code_point byte_string ')'
  code_point_list   -> CODEPOINT '/' code_point_list
                    |  EMPTY
  code_point        -> CODEPOINT
                    |  EMPTY
  byte_string       -> '(' BYTESTRING ')'
                    |  EMPTY
```

A comment behind brackets explains the reason for substitution:

```
'invalid character'
'incomplete character'
'incomplete character at the end of a record'
```

The notations STP (stop), IGN (ignore), SUB (substitute), XLT (transliteration), IDT (identical) or SKP (skip) before round brackets indicate the action taken. If the task can not be determined then UKW (unknown) is used. A ignore of an invalid character is realized as substitution with nothing. All substitutions with a length of 0 are indicated with IGN (ignore). The position of such an abnormal/irregular character is coded inside the braces:

- UNIT number of records or blocks (depends on the data)
- OFFSET byte offset up to this unit in the complete data stream
- POSITION number of bytes inside the unit where the activity was done

All counting starts with 0. The first block or record has unit 0.

The position is relative to the data stream which was presented to the character conversion module. After conversions and formatting the position values can be far off from the input file which was read originally. For example, when reading a text file with text formatting to records, then delimiting characters are not included in the byte offset, the units are records and the position indicates which character was the issue for this log entry.

The CODEPOINTs (UNICODE) are taken from the defined substitution table The BYTESTRINGs are taken from the input (read characters) and output data (written characters). BYTESTRINGs and CODEPOINTs are provided if they are known in the cases of such an irregularity. In some cases, these values can be missing.

For interpreting the report file, you must know that only the values inside round brackets are meaningful. Text before or behind round brackets are comments, but this may be subject to changes.

SELECTIONS

- `STDOUT -Write output to stdout`
- `STDERR -Write output to stderr`

4.3.97 PARAMETER *SKPBIN*

SYNOPSIS

```
HELP:   Skip character conversion if binary data is detected [FALSE]
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: SKPBIN
```

DESCRIPTION If set, this switch bypasses the character conversion module if the from-code cannot be detected. Charset detection is performed only when no CCSID or DEFAULT is provided. For automated conversion, it is useful to activate this mechanism. This may allow you to process binary and character data using the same command by performing character conversion only on printable, but not on binary data.

4.3.98 PARAMETER *SKPEQU*

SYNOPSIS

```
HELP:   Skip character conversion if from-code equals to-code [FALSE]
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: SKPEQU
```

DESCRIPTION If set, this switch bypasses the character conversion module if the from-code equals the to-code (which may happen when using charset auto detection), resulting in increased performance. This, however, also means that the input data will not be checked for invalid characters.

4.3.99 OBJECT *BAS*

SYNOPSIS

```
HELP:   Convert binary data to Base16/32/64/85 encoded data
PATH:   XCNV.OUTPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: BAS(BUFSIZ=num,RECCNT=num,METHOD=BASE16/BASE32/BASE64,MODE=BLOCK/RECORD,CHRSET=NONE ↩
    /SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/ ↩
    UTF32LE,LINE=num,DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM,CRCCHK, ↩
    ARMOR())
```

DESCRIPTION The base converter can be used to convert data from binary format to printable text and vice versa using one of a couple of base encoding schemes (RFC 4648). When writing, binary data is encoded, when reading, encoded text is decoded. Encoding data to a different base results in expansion of the data. The ratio of expansion depends on the base used.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding and decoding can be performed in one of two modes:

The block mode converts the input data as one large block of binary data. Padding will only be added at the end of the output file. This mode is the default for any type of input data, unless specified otherwise. This implies that record-based data will not retain record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* length parameter specifies after how many characters a line is wrapped. The line delimiter can be configured, but can also be omitted, which is useful to write the output record-oriented. When writing record-oriented, each output line results in one record. Note that if the *line* length parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

The record mode can be useful if the input data is record-based. In record mode, each record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. To read a file that has been encoded in record mode, setting mode=record is mandatory. Otherwise, the result might be garbage.

The character set used (ASCII or EBCDIC) can also be configured. By default, the system-specific character set is used.

Mainly for PGP support, the component supports optional ASCII armor (also in EBCDIC) headers and trailers around the base encoded data (see RFC4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified. When reading, the trailer string can optionally be verified to match the header string.

Another feature for OpenPGP support is a CRC checksum that is appended to the data. This is also optional and usable independently of the data, the encoding schema and the ASCII armor support. When writing, a switch can be enabled to add a CRC24 checksum conforming to RCF4880 for Base64 encodings. The other encoding schemas use the same mechanism to append a checksum to the base encoded data, but use another CRC variant. Base16 uses CRC32 and Base32 uses CRC40, known from the GSM network. When reading, a switch enables verification of this checksum. By default, no CRC verification is done. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be

compared against. If the verification fails, you get an error. If the verification is successful, you get an informational. If there is no checksum appended for verification, you get a warning. The same is true if verification is not requested, but a checksum is found.

ARGUMENTS

- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:RECCNT=num -Initial amount of records for preallocation [128]
- NUMBER:METHOD=BASE16/BASE32/BASE64 -Base encoding selector [BASE64]
  - BASE16 -Base16 (hexadecimal) encoding
  - BASE32 -Base32 encoding
  - BASE64 -Base64 encoding
- NUMBER:MODE=BLOCK/RECORD -Enforces record or block-oriented conversion [BLOCK]
  - BLOCK -Block-oriented conversion (all data is one chunk)
  - RECORD -Record-oriented conversion (convert each record individually)
- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:LINE=num -Maximum length of an encoded line when mode is BLOCK [0=unlimited]
- NUMBER:DELIM=HOST/BIN/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM -Line delimiter to use when mode is BLOCK and linlen>0 [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
- SWITCH:CRCCHK -Add CRC checksum to the base encoded data [FALSE]

4.3.100 OBJECT *ARMOR*

SYNOPSIS

```
HELP:   Activate ARMOR encoding and defines corresponding parameter
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.BAS
TYPE:   OBJECT
SYNTAX: ARMOR(HEADER='str',VERSION='str',COMMENT='str')
```

DESCRIPTION The ARMOR object activates ASCII armor encoding. Depending on the specified CHRSET, it may also use EBCDIC encoding. ASCII armored data is Base64 encoded binary data followed by an optional checksum, surrounded by a header and trailer. For this you can set the header, version and comment strings illustrated in the example below:

```
-----BEGIN header string-----
Version: version string
Comment: comment string

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END header string-----
```

The header data is usually set by the previous component in a meaningful way. Therefore, we recommend not overwrite these values, especially when using the PGP converter.

In general, it only makes sense to set the header data if none is set by another component (e.g. after reading binary). In this case, the default header is the current data type BINARY/RECORD/TEXT/..., the version is the current FLAM version and the comment the current default comment.

Applying ASCII armor on an XML file would yield a result similar to the example below:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10219
Comment: created with FLUCv5.1.8.10219 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
-----END XML DATA-----
```

If you activate the ARMOR object and no line length is specified, then a default of 72 is used. ASCII armor can be applied to block- or record- oriented data.

When reading, header and trailer are stripped away and the decoded data is passed to the next component. If no header is found, only the decoding is done. To verify that trailer contains the same string used in the header, you must enable a switch.

The base encoding component also supports the optional CRC24 checksum for Base64 defined in RFC4880. For the other base encoding schemas, CRC checksum lengths are used that match the respective block size.

The same XML example as above, but with a CRC24 checksum, looks like this:

```
-----BEGIN XML DATA-----
Version: FLAM: 5.1.8.10233
Comment: created with FLUCv5.1.8.10233 (www.flam.de)
Charset: UTF-8

PD94bWwgdmVyc2lvbj0iMS4wIj8+CjwhRE9DVFlQRSB0ZXN0IFsKPCFFTEVNRU5UIHRlc3Qg
KCNQQ0RBVEEpPgo8IUVOVElUWSBhYSAiJiMzNzt6ejsmYW1wO2FhOyI+CjwhRU5USVRZICUg
```

```
eHggIiYjMzc7eno7Ij4KPCFFTlRJVFkgJSB6eiAiPCFFTlRJVFkgdHJpY2t5ICYjMzQ7ZXJy
b3ItcHJvbmUmIzM0OyA+Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgU1lTVEVNICJzeXN0
ZW1JZCV6ejsmYW1wOyYjMzQ7Ij4KPCFOT1RBVElPTiBub3RhdGlvbk5hbWUgUFVCTElDICJw
dWJsaWNJZCIgInN5c3RlbUlkJiMzNDsiPgoleHg7CiV4eDsKXT4KPHRlc3Q+VGhpcyBzYW1w
bGUgc2hvd3MgYSAmdHJpY2t5OyBtZXRob2QuPC90ZXN0Pg==
=3m94
-----END XML DATA-----
```

To verify the checksum, a switch must be enabled as it is not verified by default. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against.

ATTENTION An empty file that has been ASCII-armored consists of a header and trailer plus optional CRC checksum (00...00), only. Such a file is not detected automatically as ASCII-armored when trying to read it. It will be handled as normal text file with delimiters. To read such a file as base encoded without any data, you must set the correct decoding method.

```
-----BEGIN EMPTY FILE-----
Version: TEST VERSION
Comment: TEST COMMENT


=00000000
-----END EMPTY FILE-----
```

From the perspective of auto detection, such a file is a simple text file. Files are only detected as ASCII-armored if they contain some actual data.

ARGUMENTS

- `STRING:HEADER='str' -String for the header [AUTO]`
- `STRING:VERSION='str' -Version string as key value [AUTO]`
- `STRING:COMMENT='str' -Comment string as key value [AUTO]`

4.3.101 OBJECT *HSH*

SYNOPSIS

```
HELP:   Generation or verification of one way hash values
PATH:   XCNV.OUTPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: HSH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/ ↩
    CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.3.102 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   XCNV.OUTPUT.SAV.FILE.CNV.HSH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                              ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'                  ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'                 ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                    ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'         ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.3.103 OVERLAY *FIO*

SYNOPSIS

```
HELP:   File IO [BLK]
PATH:   XCNV.OUTPUT.SAV.FILE
TYPE:   OVERLAY
SYNTAX: FIO.{BLK()/REC()/TXT()/FL4()}
```

DESCRIPTION With the overlay FIO you can choose different kind of file IO methods for write operations.

4.3.104 OBJECT *BLK*

SYNOPSIS

```
HELP:   Write a file block-oriented (mainly open world)
PATH:   XCNV.OUTPUT.SAV.FILE.FIO
TYPE:   OBJECT
SYNTAX: BLK(NAME='str'/STREAM/DUMMY,FALLOC(),REMAIN,APPEND,FLUSH,FRCREC,ADDDLM,ACSTIM=num, ↩
    MODTIM=num,PRNCTR=DETACH/RETAIN/ERASE,BINDMP)
```

DESCRIPTION This object defines the block-oriented write operation for files. This method writes the all data after any conversions as one chunk to the file. This is the most common and simplest way to write a file. This write method is useful for all non-record-oriented file formats.

If the block still contains a record length information, you can enforce a record-oriented write. It is also possible to add the system delimiter behind each record to produce a valid text block.

ARGUMENTS

- STRING:NAME='str'/STREAM/DUMMY -Name/URL of file to write ["==stdout]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- SWITCH:REMAIN -Remain existing files (don't overwrite) [FALSE]
- SWITCH:APPEND -Append data to this file [FALSE]
- SWITCH:FLUSH -Enforce flush in front of close file [FALSE]
- SWITCH:FRCREC -Enforce record orientation on record oriented devices [FALSE]
- SWITCH:ADDDLM -Add a delimiter behind each record to produce a text block [FALSE]
- NUMBER:ACSTIM=num -Define the last access time for this file
- NUMBER:MODTIM=num -Define the last modification time for this file
- SWITCH:BINDMP -Write dump for binary data [FALSE]

4.3.105 OBJECT *FALLOC*

SYNOPSIS

```
HELP:   Platform dependent file allocation parameter
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.BLK
TYPE:   OBJECT
SYNTAX: FALLOC(ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZE=num,RECFORMAT=NONE/BIN/TXT/ ↩
    DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX- ↩
    REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR ↩
    /FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/ ↩
    VRA/VRM,RECLENGTH=num,KEYDISP=OLD/NEW/DEL,KEYPOSITION=num,KEYLENGTH=num,SUBSYSTEM(), ↩
    SPACE(),LIKE='str',RENEW)
```

DESCRIPTION The file allocation object collects several parameters for platforms which are record oriented based on a catalog system like z/OS® and where the space, volume and other things must be predefined. The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

But also on distributed platforms like (WIN, UNIX, MAC, ...) FLAM supports record oriented file types to keep the record information on such platforms. For this case the different record format for the open world can be defined. To read such a special file format on an open platform the correct record format must also be defined. On a catalog based system, this is not required. If the record format in the catalog are not the same as specified an error will occur.

ARGUMENTS

- NUMBER:ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [SEQ]
  - SEQ -Sequential data set [DEFAULT]
  - LIB -Library (ZOS-PDSE)
  - PDS -Partitioned data set (ZOS-PDS)
  - USS -UNIX files (path based)
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)
- NUMBER:BLKSIZE=num -Block size [system]
- NUMBER:RECFORMAT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format [auto]
  - NONE -No format defined
  - BIN -Open -Binary (undefined) data (only the last record are shorter then record length)
  - TXT -Open -Text data with standard delimiter (result is a stripped variable length records)
  - DLM -Open -Records separated by a special delimiter with the result of variable length records
  - VAR -Open -Variable length record (if necessary define length format)
  - VAR-ASA -Open -Variable length record with ASA print control character (if necessary define length format)
  - VAR-MCC -Open -Variable length record with machine print control codes (if necessary define length format)
  - FIX -Open -Fix length record (file size =N * record length)

- FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
- FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
- VAR-REL -Open -Variable relative record (if necessary define length format)
- VAR-REL-ASA -Open -Variable relative record with ASA print control character (if ne cessary define length format)
- VAR-REL-MCC -Open -Variable relative record with machine print control codes (if ne cessary define length format)
- FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
- FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size = N * (record length + 9))
- FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size = N * (record length + 9))
- F -Host -Fixed-length, unblocked
- FA -Host -Fixed-length, unblocked, ASA print control characters
- FM -Host -Fixed-length, unblocked, machine print control codes
- FS -Host -Fixed-length, standard
- FSA -Host -Fixed-length, standard, ASA print control characters
- FSM -Host -Fixed-length, standard, machine print control codes
- FB -Host -Fixed-length, blocked
- FBA -Host -Fixed-length, blocked, ASA print control characters
- FBM -Host -Fixed-length, blocked, machine print control codes
- FBS -Host -Fixed-length, blocked, standard
- FBSA -Host -Fixed-length, blocked, standard, ASA print control characters
- FBSM -Host -Fixed-length, blocked, standard, machine print control codes
- FE -Host -Fixed-length, ESDS
- FEA -Host -Fixed-length, ESDS, ASA print control characters
- FEM -Host -Fixed-length, ESDS, machine print control codes
- FK -Host -Fixed-length, KSDS
- FKA -Host -Fixed-length, KSDS, ASA print control characters
- FKM -Host -Fixed-length, KSDS, machine print control codes
- FR -Host -Fixed-length, RRDS
- FRA -Host -Fixed-length, RRDS, ASA print control characters
- FRM -Host -Fixed-length, RRDS, machine print control codes
- U -Host -Undefined-length
- UA -Host -Undefined-length, ASA print control characters
- UM -Host -Undefined-length, machine print control codes
- V -Host -Variable
- VA -Host -Variable, ASA print control characters
- VM -Host -Variable, machine print control codes
- VS -Host -Variable, spanned
- VSA -Host -Variable, spanned, ASA print control characters
- VSM -Host -Variable, spanned, machine print control codes
- VB -Host -Variable, blocked

  - VBA -Host -Variable, blocked, ASA print control characters
  - VBM -Host -Variable, blocked, machine print control codes
  - VBS -Host -Variable, blocked, spanned
  - VBSA -Host -Variable, blocked, spanned, ASA print control characters
  - VBSM -Host -Variable, blocked, spanned, machine print control codes
  - VE -Host -Variable, ESDS
  - VEA -Host -Variable, ESDS, ASA print control characters
  - VEM -Host -Variable, ESDS, machine print control codes
  - VK -Host -Variable, KSDS
  - VKA -Host -Variable, KSDS, ASA print control characters
  - VKM -Host -Variable, KSDS, machine print control codes
  - VR -Host -Variable, RRDS
  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes
- NUMBER:RECLENGTH=num -Record length (data only, no print control and length fields)
- NUMBER:KEYDISP=OLD/NEW/DEL -Key disposition for index/key sequential data sets [OLD]
  - OLD -Key disposition OLD (keep the existing key)
  - NEW -Key disposition NEW (insert record number as key)
  - DEL -Key disposition DEL (delete key from record)
- NUMBER:KEYPOSITION=num -Key position (>=1) for index/key sequential data sets [1]
- NUMBER:KEYLENGTH=num -Key length (>=1) for index/key sequential data sets [8]
- STRING:LIKE='str' -Specifies the name of the model data set from which the attributes are to be copied
- SWITCH:RENEW -Enforce new allocation by a remove of an existing file

4.3.106 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- STRING:NAME='str' -Name of the subsystem
- STRING:PARM='str' -Parameter list for the subsystem (comma separated)

4.3.107 OBJECT *SPACE*

SYNOPSIS

```
HELP:   Platform dependent space definition parameter
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC
TYPE:   OBJECT
SYNTAX: SPACE(PRIMARY=num,SECONDARY=num,DIRBLOCKS=num,VOLSER='str',VOLUNIT='str',DATACLASS ↩
    ='str',MGMTCLASS='str',STORCLASS='str',NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG, ↩
    ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG,RELEASE=YES/NO,UNMOVABLE,CONTIG,PERMANENT,RLS= ↩
    NRI/CR/CRE)
```

DESCRIPTION The space definition object collects several parameters for platforms where data sets are preallocated like z/OS® and where clylinders or tracks must be predefined for a file allocation.

The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

All size values must be given in megabyte (MiB). The calculation in the platform dependent units (tracks, cylinders, ...) are done inernally. Note: 1 MiB = 1024 KiB = 1024*1024 Byte

ARGUMENTS

- NUMBER:PRIMARY=num -Primary size in megabytes
- NUMBER:SECONDARY=num -Secondary size in megabytes
- NUMBER:DIRBLOCKS=num -Directory blocks for partitioned data sets
- STRING:VOLSER='str' -Volume serial number
- STRING:VOLUNIT='str' -Volume unit
- STRING:DATACLASS='str' -Data class
- STRING:MGMTCLASS='str' -Management class
- STRING:STORCLASS='str' -Storage class
- NUMBER:NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for normal end [CATALOG]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for abnormal end [DELETE]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete

  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:RELEASE=YES/NO -Release unused space when file is closed [SEQ=YES, PDS=NO]
  - YES -Yes, do it
  - NO -No, don't do it
- SWITCH:UNMOVABLE -Allocate space unmovable [FALSE]
- SWITCH:CONTIG -Allocate space contiguously [FALSE]
- SWITCH:PERMANENT -Allocate space permanently [FALSE]
- NUMBER:RLS=NRI/CR/CRE -Specifies the type of record level sharing (RLS) being done for a specific data set [NONE]
  - NRI -No Read Integrity
  - CR -Consistent Read
  - CRE -Consistent Read Explicit

4.3.108 PARAMETER *PRNCTR*

SYNOPSIS

```
HELP:   Print control character handling if write ASA or MCC records [DETACH]
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.BLK
TYPE:   NUMBER
SYNTAX: PRNCTR=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control character handling is a special feature for record oriented files of mainframe system. For print control, the ASA or machine control charaters are encoded in the first byte of the record.

If an attribute with such a control character is known on write operation, you can decide whether it will be part of the output (RETAIN), written to the output if required (DETACH) or never written (ERASE).

SELECTIONS

- DETACH -Manage print control character as record attribute
- RETAIN -Leave print control character as part of the data record
- ERASE -Erase the print control character from the record

4.3.109 OBJECT *REC*

SYNOPSIS

```
HELP:   Write a file record-oriented (mainly mainframe world)
PATH:   XCNV.OUTPUT.SAV.FILE.FIO
TYPE:   OBJECT
SYNTAX: REC(NAME='str'/STREAM/DUMMY,FALLOC(),REMAIN,APPEND,FLUSH,RECMOD=STOP/CUT/WRAP, ↩
    PRNCTR=DETACH/RETAIN/ERASE,RECCNT=num,LENFMT.{},SUPPAD,PADCHR='bin'/BINARY-ZERO/ASCII- ↩
    BLANK/EBCDIC-BLANK/UTF08-BLANK/UTF16BE-BLANK/UTF16LE-BLANK/UTF32BE-BLANK/UTF32LE-BLANK, ↩
    CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/ ↩
    UCS4LE/UTF32LE,RECDLM='bin'/CR-ASCII/LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF-EBCDIC/NL ↩
    -EBCDIC/CRLF-EBCDIC/CR-UTF08/LF-UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE/NL- ↩
    UTF16BE/CRLF-UTF16BE/CR-UTF16LE/LF-UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF-UTF32BE ↩
    /NL-UTF32BE/CRLF-UTF32BE/CR-UTF32LE/LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE,ACSTIM=num,MODTIM ↩
    =num)
```

DESCRIPTION This object defines the record-oriented write operation for files. This method writes each data list entry as one record to a data set. You can define the behavior (STOP/CUT/WRAP) if the record is longer than the accepted record length for this file. You can activate the suppression of padding characters and set the padding character. You can define the file organization, block size, record format, add a delimiter behind each record and a lot of other parameters (see below).

A dynamic allocation of the file can be used. This is very powerful and simplifies the generation of files on mainframe systems. The default values for this dynamic allocation depend on the data itself and the original file where the data came from. For example, if you have read an FB80 data set with a GZIP data stream which was formatted based on delimiters in text records and you write it to a file on z/OS without any further parameters, then a PS-VB data set (not FB) with data dependent records is allocated with the correct size. If the result of the decompression is still binary data, then still a PS-FB80 dataset will be allocated. But you can also define other parameters for the file allocation. In this case, the data is converted to this format as good as possible.

On non-mainframe platforms (Windows, Unix, ...), the record I/O module can be used to write records terminated by a delimiter (TXT/DLM) or in a fixed or variable length format. For variable format, the length format must be defined. The default in this case is a 4 byte integer in the platform-dependent endianess. This default conforms to FILEDATA=RECORD for USS on a z/OS mainframe system.

If you chose a host record format for the write on a non-record-oriented Attributes (length, slot number of RRDS, print control character) are stored in front of each record on non-record-oriented systems. Example: If you read a RRDS with print control characters (FRA) and you write this on UNIX as record format VRA to a file, the file format looks like this:

If you chose a host record format for writing on a non-record-oriented system, then the length (only if it is a variable format) and other attributes (slot number, print control character) are stored in front of the data. Here is an example for VRA (variable, relative record format with ASA print control character):

```
LLLLnnnnnnnnCdddd..dddd
LLLLnnnnnnnnCdddd..dddd
LLLLnnnnnnnnCdddd..dddd
...
...
...
LLLLnnnnnnnnCdddd..dddd
LLLLnnnnnnnnCdddd..dddd
```

Where LLLL is the record length, nn...nn is the slot number, C is the print control character and dd..dd is the record data. To read such a file on the UNIX system, the record format and length format (if you write with defaults, you can read with defaults) must be defined correctly.

This feature allows to unload an RRDS to a WINDOWS or UNIX system and to load this RRDS again with same records and gaps between. Each supported record format has a corresponding representation on non-mainframe systems. This means this unload and load is possible for each kind of data set type of mainframe systems.

ARGUMENTS

- STRING:NAME='str'/STREAM/DUMMY -Name/URL of file to write ["==stdout]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- SWITCH:REMAIN -Remain existing files (don't overwrite) [FALSE]
- SWITCH:APPEND -Append data to this file [FALSE]
- SWITCH:FLUSH -Enforce flush in front of close file [FALSE]
- NUMBER:RECMOD=STOP/CUT/WRAP -Mode used to write records [STOP]
  - STOP -Stop if record too long
  - CUT -Cut if record too long
  - WRAP -Wrap if record too long

- NUMBER:RECCNT=num -Initial amount of records for preallocation [128]
- SWITCH:SUPPAD -Suppress trailing padding characters (useful for variable length reco rds)
- STRING:PADCHR='bin'/BINARY-ZERO/ASCII-BLANK/EBCDIC-BLANK/UTF08-BLANK/UTF16BE-BLANK/UTF 16LE-BLANK/UTF32BE-BLANK/UTF32LE-BLANK -Character to pad records to a fix length
  - BINARY-ZERO -Padding with 0x00
  - ASCII-BLANK -Padding with 0x20
  - EBCDIC-BLANK -Padding with 0x40
  - UTF08-BLANK -Padding with 0x20
  - UTF16BE-BLANK -Padding with 0x0020
  - UTF16LE-BLANK -Padding with 0x2000
  - UTF32BE-BLANK -Padding with 0x00000020
  - UTF32LE-BLANK -Padding with 0x20000000
- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/ UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- STRING:RECDLM='bin'/CR-ASCII/LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF-EBCDIC/NL-EBC DIC/CRLF-EBCDIC/CR-UTF08/LF-UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE/NL-UTF16BE/ CRLF-UTF16BE/CR-UTF16LE/LF-UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF-UTF32BE/NL-UTF32BE/CRLF-UTF32BE/CR-UTF32LE/LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE -Delimiter used t o define end of record
  - CR-ASCII -Delimiter:carriage return in ASCII (0x0D)
  - LF-ASCII -Delimiter:line feed in ASCII (0x0A)
  - NL-ASCII -Delimiter:new line in ASCII (0x85)
  - CRLF-ASCII -Delimiter:carriage return line feed in ASCII (0x0D0A)
  - CR-EBCDIC -Delimiter:carriage return in EBCDIC (0x0D)
  - LF-EBCDIC -Delimiter:line feed in EBCDIC (0x25)
  - NL-EBCDIC -Delimiter:new line in EBCDIC (0x15)
  - CRLF-EBCDIC -Delimiter:carriage return line feed in EBCDIC (0x0D25)
  - CR-UTF08 -Delimiter:carriage return in UTF-8 (0x0D)

  - LF-UTF08 -Delimiter:line feed in UTF-8 (0x0A)
  - NL-UTF08 -Delimiter:new line in UTF-8 (0xC285)
  - CRLF-UTF08 -Delimiter:carriage return line feed in UTF-8 (0x0D0A)
  - CR-UTF16BE -Delimiter:carriage return in UTF-16BE (0x000D)
  - LF-UTF16BE -Delimiter:line feed in UTF-16BE (0x000A)
  - NL-UTF16BE -Delimiter:new line in UTF-16BE (0x0085)
  - CRLF-UTF16BE -Delimiter:carriage return line feed in UTF-16BE (0x000D000A)
  - CR-UTF16LE -Delimiter:carriage return in UTF-16LE (0x0D00)
  - LF-UTF16LE -Delimiter:line feed in UTF-16LE (0x0A00)
  - NL-UTF16LE -Delimiter:new line in UTF-16LE (0x8500)
  - CRLF-UTF16LE -Delimiter:carriage return line feed in UTF-16LE (0x0D000A00)
  - CR-UTF32BE -Delimiter:carriage return in UTF-32BE (0x0000000D)
  - LF-UTF32BE -Delimiter:line feed in UTF-32BE (0x0000000A)
  - NL-UTF32BE -Delimiter:new line in UTF-32BE (0x00000085)
  - CRLF-UTF32BE -Delimiter:carriage return line feed in UTF-32BE (0x0000000D0000000A)
  - CR-UTF32LE -Delimiter:carriage return in UTF-32LE (0x0D000000)
  - LF-UTF32LE -Delimiter:line feed in UTF-32LE (0x0A000000)
  - NL-UTF32LE -Delimiter:new line in UTF-32LE (0x85000000)
  - CRLF-UTF32LE -Delimiter:carriage return line feed in UTF-32LE (0x0D0000000A000000)

- NUMBER:ACSTIM=num -Define the last access time for this file
- NUMBER:MODTIM=num -Define the last modification time for this file

4.3.110 OBJECT *FALLOC*

SYNOPSIS

```
HELP:   Platform dependent file allocation parameter
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.REC
TYPE:   OBJECT
SYNTAX: FALLOC(ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZE=num,RECFORMAT=NONE/BIN/TXT/ ↩
    DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX- ↩
    REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR ↩
    /FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/ ↩
    VRA/VRM,RECLENGTH=num,KEYDISP=OLD/NEW/DEL,KEYPOSITION=num,KEYLENGTH=num,SUBSYSTEM(), ↩
    SPACE(),LIKE='str',RENEW)
```

DESCRIPTION The file allocation object collects several parameters for platforms which are record oriented based on a catalog system like z/OS® and where the space, volume and other things must be predefined. The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

But also on distributed platforms like (WIN, UNIX, MAC, ...) FLAM supports record oriented file types to keep the record information on such platforms. For this case the different record format for the open world can be defined. To read such a special file format on an open platform the correct record format must also be defined. On a catalog based system, this is not required. If the record format in the catalog are not the same as specified an error will occur.

ARGUMENTS

- NUMBER:ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [SEQ]
  - SEQ -Sequential data set [DEFAULT]

  - LIB -Library (ZOS-PDSE)
  - PDS -Partitioned data set (ZOS-PDS)
  - USS -UNIX files (path based)
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)
- NUMBER:BLKSIZE=num -Block size [system]
- NUMBER:RECFORMAT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format [auto]
  - NONE -No format defined
  - BIN -Open -Binary (undefined) data (only the last record are shorter then record length)
  - TXT -Open -Text data with standard delimiter (result is a stripped variable length records)
  - DLM -Open -Records separated by a special delimiter with the result of variable length records
  - VAR -Open -Variable length record (if necessary define length format)
  - VAR-ASA -Open -Variable length record with ASA print control character (if necessary define length format)
  - VAR-MCC -Open -Variable length record with machine print control codes (if necessary define length format)
  - FIX -Open -Fix length record (file size =N * record length)
  - FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
  - FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
  - VAR-REL -Open -Variable relative record (if necessary define length format)
  - VAR-REL-ASA -Open -Variable relative record with ASA print control character (if necessary define length format)
  - VAR-REL-MCC -Open -Variable relative record with machine print control codes (if necessary define length format)
  - FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
  - FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size =N * (record length + 9))
  - FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size =N * (record length + 9))
  - F -Host -Fixed-length, unblocked
  - FA -Host -Fixed-length, unblocked, ASA print control characters
  - FM -Host -Fixed-length, unblocked, machine print control codes
  - FS -Host -Fixed-length, standard
  - FSA -Host -Fixed-length, standard, ASA print control characters
  - FSM -Host -Fixed-length, standard, machine print control codes
  - FB -Host -Fixed-length, blocked
  - FBA -Host -Fixed-length, blocked, ASA print control characters

  - FBM -Host -Fixed-length, blocked, machine print control codes
  - FBS -Host -Fixed-length, blocked, standard
  - FBSA -Host -Fixed-length, blocked, standard, ASA print control characters
  - FBSM -Host -Fixed-length, blocked, standard, machine print control codes
  - FE -Host -Fixed-length, ESDS
  - FEA -Host -Fixed-length, ESDS, ASA print control characters
  - FEM -Host -Fixed-length, ESDS, machine print control codes
  - FK -Host -Fixed-length, KSDS
  - FKA -Host -Fixed-length, KSDS, ASA print control characters
  - FKM -Host -Fixed-length, KSDS, machine print control codes
  - FR -Host -Fixed-length, RRDS
  - FRA -Host -Fixed-length, RRDS, ASA print control characters
  - FRM -Host -Fixed-length, RRDS, machine print control codes
  - U -Host -Undefined-length
  - UA -Host -Undefined-length, ASA print control characters
  - UM -Host -Undefined-length, machine print control codes
  - V -Host -Variable
  - VA -Host -Variable, ASA print control characters
  - VM -Host -Variable, machine print control codes
  - VS -Host -Variable, spanned
  - VSA -Host -Variable, spanned, ASA print control characters
  - VSM -Host -Variable, spanned, machine print control codes
  - VB -Host -Variable, blocked
  - VBA -Host -Variable, blocked, ASA print control characters
  - VBM -Host -Variable, blocked, machine print control codes
  - VBS -Host -Variable, blocked, spanned
  - VBSA -Host -Variable, blocked, spanned, ASA print control characters
  - VBSM -Host -Variable, blocked, spanned, machine print control codes
  - VE -Host -Variable, ESDS
  - VEA -Host -Variable, ESDS, ASA print control characters
  - VEM -Host -Variable, ESDS, machine print control codes
  - VK -Host -Variable, KSDS
  - VKA -Host -Variable, KSDS, ASA print control characters
  - VKM -Host -Variable, KSDS, machine print control codes
  - VR -Host -Variable, RRDS
  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes
- NUMBER:RECLENGTH=num -Record length (data only, no print control and length fields)
- NUMBER:KEYDISP=OLD/NEW/DEL -Key disposition for index/key sequential data sets [OLD]
  - OLD -Key disposition OLD (keep the existing key)
  - NEW -Key disposition NEW (insert record number as key)
  - DEL -Key disposition DEL (delete key from record)
- NUMBER:KEYPOSITION=num -Key position (>=1) for index/key sequential data sets [1]
- NUMBER:KEYLENGTH=num -Key length (>=1) for index/key sequential data sets [8]
- STRING:LIKE='str' -Specifies the name of the model data set from which the attributes are to be copied
- SWITCH:RENEW -Enforce new allocation by a remove of an existing file

4.3.111 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.3.112 OBJECT *SPACE*

SYNOPSIS

```
HELP:   Platform dependent space definition parameter
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC
TYPE:   OBJECT
SYNTAX: SPACE(PRIMARY=num,SECONDARY=num,DIRBLOCKS=num,VOLSER='str',VOLUNIT='str',DATACLASS ↩
    ='str',MGMTCLASS='str',STORCLASS='str',NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG, ↩
    ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG,RELEASE=YES/NO,UNMOVABLE,CONTIG,PERMANENT,RLS= ↩
    NRI/CR/CRE)
```

DESCRIPTION The space definition object collects several parameters for platforms where data sets are preallocated like z/OS® and where clylinders or tracks must be predefined for a file allocation.

The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

All size values must be given in megabyte (MiB). The calculation in the platform dependent units (tracks, cylinders, ...) are done inernally. Note: 1 MiB = 1024 KiB = 1024*1024 Byte

ARGUMENTS

- `NUMBER:PRIMARY=num -Primary size in megabytes`
- `NUMBER:SECONDARY=num -Secondary size in megabytes`
- `NUMBER:DIRBLOCKS=num -Directory blocks for partitioned data sets`

- STRING:VOLSER='str' -Volume serial number
- STRING:VOLUNIT='str' -Volume unit
- STRING:DATACLASS='str' -Data class
- STRING:MGMTCLASS='str' -Management class
- STRING:STORCLASS='str' -Storage class
- NUMBER:NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for normal end [CATALOG]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for abnormal end [DELETE]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:RELEASE=YES/NO -Release unused space when file is closed [SEQ=YES, PDS=NO]
  - YES -Yes, do it
  - NO -No, don't do it
- SWITCH:UNMOVABLE -Allocate space unmovable [FALSE]
- SWITCH:CONTIG -Allocate space contiguously [FALSE]
- SWITCH:PERMANENT -Allocate space permanently [FALSE]
- NUMBER:RLS=NRI/CR/CRE -Specifies the type of record level sharing (RLS) being done for a specific data set [NONE]
  - NRI -No Read Integrity
  - CR -Consistent Read
  - CRE -Consistent Read Explicit

4.3.113 PARAMETER *PRNCTR*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.REC
TYPE:   NUMBER
SYNTAX: PRNCTR=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control character handling is a special feature for record oriented files of mainframe system. For print control, the ASA or machine control charaters are encoded in the first byte of the record.

If an attribute with such a control character is known on write operation, you can decide whether it will be part of the output (RETAIN), written to the output if required (DETACH) or never written (ERASE).

SELECTIONS

- DETACH -Manage print control character as record attribute
- RETAIN -Leave print control character as part of the data record
- ERASE -Erase the print control character from the record

4.3.114 OVERLAY *LENFMT*

SYNOPSIS

```
HELP:   Format of the length field for open platforms
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.REC
TYPE:   OVERLAY
SYNTAX: LENFMT.{INTEGER()/STRING()/BCD()/HOST()}
```

DESCRIPTION The overlay can be used to define the format of the length filed on open platforms which don't support record formats by the data management system. For example: If you read records on a mainframe and you want to write the records as records (not as binary or text data) on WINDOWS or UNIX then the data will be prefixed with the record length. The format of this length field can be changed over this union.

This length specification will be only used for variable length records. The default mapping method between record formats will be map each host record format the corresponding open variable format.

If the length format not defined the length field will be written as 32 bit integer in platform dependent byte order. This behavior corresponds to the FILEDATA=RECORD parameter for USS files on z/OS. Means if you write a record oriented file to USS without any format changed over this overlay then you can read the record with another MVS application using FILEDATA=RECORD.

If you choose record format text or binary then this will correspond to FILEDATA=TEXT or FILEDATA=BINARY and the length format specification will be ignored. To enable record format mapping based on the data type you can set the environment variable below:

```
FL_RECORD_FORMAT_MAPPING=DATATYPE
```

If this environment variable set, then the length format is only used if you explicitly set a variable record format of the open systems.

Since version 5.1.8, FLAM can detect the 4 byte length fields (integer or host) inside a data stream. We recommend to use one of these formats to have more comfort when reading.

4.3.115 OBJECT *INTEGER*

SYNOPSIS

```
HELP:   Use integer format for the length field
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT
TYPE:   OBJECT
SYNTAX: INTEGER(ENDIAN=SYSTEM/BIG/LITTLE,WIDTH=U16/U32/U64,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in integer format. You can define the byte order, the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- `NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [SYSTEM]`
  - `SYSTEM -Byte order of the current system`
  - `BIG -Big endian byte order`
  - `LITTLE -Little endian byte order`
- `NUMBER:WIDTH=U16/U32/U64 -Bit width/rate [B32]`
  - `U16 -16 bit unsigned integer`
  - `U32 -32 bit unsigned integer`
  - `U64 -64 bit unsigned integer`
- `SWITCH:INCLUDE -Include length field in length value [OFF]`

4.3.116 OBJECT *STRING*

SYNOPSIS

```
HELP:   Use string format for the length field
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT
TYPE:   OBJECT
SYNTAX: STRING(CHRSET=SYSTEM/ASCII/EBCDIC,COUNT=C04/C05/C06,BASE=B10,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in string format (a character per digit). You can define the character set, the character count, the base and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:CHRSET=SYSTEM/ASCII/EBCDIC -Character set [SYSTEM]
  - SYSTEM -Use system code page
  - ASCII -Use ASCII (UTF-8) code page
  - EBCDIC -Use EBCDIC code page
- NUMBER:COUNT=C04/C05/C06 -Character count [C06]
  - C04 -4 character/digits
  - C05 -5 character/digits
  - C06 -6 character/digits
- NUMBER:BASE=B10 -String base [B10]
  - B10 -Base 10 (decimal)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.3.117 OBJECT *BCD*

SYNOPSIS

```
HELP:   Use BCD format for the length field
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT
TYPE:   OBJECT
SYNTAX: BCD(WIDTH=U16/U24/U32,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in BCD format (4 Bit are one decimal digit). You can define the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:WIDTH=U16/U24/U32 -Bit width/rate [B32]
  - U16 -16 bit unsigned BCD (POV with 4 digits)
  - U24 -24 bit unsigned BCD (POV with 6 digits)
  - U32 -32 bit unsigned BCD (POV with 8 digits)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.3.118 OBJECT *HOST*

SYNOPSIS

```
HELP:   Use host format (LLxx) for the length field
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT
TYPE:   OBJECT
SYNTAX: HOST(ENDIAN=SYSTEM/BIG/LITTLE,EXCLUDE)
```

DESCRIPTION The object defines the 4 byte host specific format (LLxx) of the length field. This format uses the first 16 bits as integer in big endian or little endian byte order to represent the record length including or excluding the record length field itself and the last 16 bits are set to zero when writing and ignored when reading.

For example, a record with 136 bytes content has a default length field (big endian, inclusive) of:

```
   x'008D0000'
```

The default conforms to the length fields used by the data management system of MVS.

ARGUMENTS

- NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [BIG ENDIAN] for first 16 Bit
  - SYSTEM -Byte order of the current system
  - BIG -Big endian byte order
  - LITTLE -Little endian byte order
- SWITCH:EXCLUDE -Record length field not part of the length value [OFF]

4.3.119 OBJECT *TXT*

SYNOPSIS

```
HELP:   Write a file text-oriented (mainly open world)
PATH:   XCNV.OUTPUT.SAV.FILE.FIO
TYPE:   OBJECT
SYNTAX: TXT(NAME='str'/STREAM/DUMMY,FALLOC(),REMAIN,APPEND,FLUSH,ACSTIM=num,MODTIM=num)
```

DESCRIPTION This object defines the text-oriented write operation for files. This method writes each data list entry as a text record with delimiter to the file. This method supposes that the data elements are text records and writes each record with the system specific delimiter (0x0A on UNIX, 0x0D0A on WINDOWS, 0x15 on USS) to a file.

ARGUMENTS

- STRING:NAME='str'/STREAM/DUMMY -Name/URL of file to write ["==stdout]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- SWITCH:REMAIN -Remain existing files (don't overwrite) [FALSE]
- SWITCH:APPEND -Append data to this file [FALSE]
- SWITCH:FLUSH -Enforce flush in front of close file [FALSE]
- NUMBER:ACSTIM=num -Define the last access time for this file
- NUMBER:MODTIM=num -Define the last modification time for this file

4.3.120 OBJECT *FALLOC*

SYNOPSIS

```
HELP:   Platform dependent file allocation parameter
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.TXT
TYPE:   OBJECT
SYNTAX: FALLOC(ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZE=num,RECFORMAT=NONE/BIN/TXT/ ↩
    DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX- ↩
    REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR ↩
    /FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/ ↩
    VRA/VRM,RECLENGTH=num,KEYDISP=OLD/NEW/DEL,KEYPOSITION=num,KEYLENGTH=num,SUBSYSTEM(), ↩
    SPACE(),LIKE='str',RENEW)
```

DESCRIPTION The file allocation object collects several parameters for platforms which are record oriented based on a catalog system like z/OS® and where the space, volume and other things must be predefined. The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

But also on distributed platforms like (WIN, UNIX, MAC, ...) FLAM supports record oriented file types to keep the record information on such platforms. For this case the different record format for the open world can be defined. To read such a special file format on an open platform the correct record format must also be defined. On a catalog based system, this is not required. If the record format in the catalog are not the same as specified an error will occur.

ARGUMENTS

- NUMBER:ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [SEQ]
  - SEQ -Sequential data set [DEFAULT]
  - LIB -Library (ZOS-PDSE)
  - PDS -Partitioned data set (ZOS-PDS)
  - USS -UNIX files (path based)
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)
- NUMBER:BLKSIZE=num -Block size [system]
- NUMBER:RECFORMAT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format [auto]
  - NONE -No format defined
  - BIN -Open -Binary (undefined) data (only the last record are shorter then record length)
  - TXT -Open -Text data with standard delimiter (result is a stripped variable length records)
  - DLM -Open -Records separated by a special delimiter with the result of variable length records
  - VAR -Open -Variable length record (if necessary define length format)
  - VAR-ASA -Open -Variable length record with ASA print control character (if necessary define length format)
  - VAR-MCC -Open -Variable length record with machine print control codes (if necessary define length format)
  - FIX -Open -Fix length record (file size =N * record length)

- FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
- FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
- VAR-REL -Open -Variable relative record (if necessary define length format)
- VAR-REL-ASA -Open -Variable relative record with ASA print control character (if ne cessary define length format)
- VAR-REL-MCC -Open -Variable relative record with machine print control codes (if ne cessary define length format)
- FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
- FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size = N * (record length + 9))
- FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size = N * (record length + 9))
- F -Host -Fixed-length, unblocked
- FA -Host -Fixed-length, unblocked, ASA print control characters
- FM -Host -Fixed-length, unblocked, machine print control codes
- FS -Host -Fixed-length, standard
- FSA -Host -Fixed-length, standard, ASA print control characters
- FSM -Host -Fixed-length, standard, machine print control codes
- FB -Host -Fixed-length, blocked
- FBA -Host -Fixed-length, blocked, ASA print control characters
- FBM -Host -Fixed-length, blocked, machine print control codes
- FBS -Host -Fixed-length, blocked, standard
- FBSA -Host -Fixed-length, blocked, standard, ASA print control characters
- FBSM -Host -Fixed-length, blocked, standard, machine print control codes
- FE -Host -Fixed-length, ESDS
- FEA -Host -Fixed-length, ESDS, ASA print control characters
- FEM -Host -Fixed-length, ESDS, machine print control codes
- FK -Host -Fixed-length, KSDS
- FKA -Host -Fixed-length, KSDS, ASA print control characters
- FKM -Host -Fixed-length, KSDS, machine print control codes
- FR -Host -Fixed-length, RRDS
- FRA -Host -Fixed-length, RRDS, ASA print control characters
- FRM -Host -Fixed-length, RRDS, machine print control codes
- U -Host -Undefined-length
- UA -Host -Undefined-length, ASA print control characters
- UM -Host -Undefined-length, machine print control codes
- V -Host -Variable
- VA -Host -Variable, ASA print control characters
- VM -Host -Variable, machine print control codes
- VS -Host -Variable, spanned
- VSA -Host -Variable, spanned, ASA print control characters
- VSM -Host -Variable, spanned, machine print control codes
- VB -Host -Variable, blocked

  - VBA -Host -Variable, blocked, ASA print control characters
  - VBM -Host -Variable, blocked, machine print control codes
  - VBS -Host -Variable, blocked, spanned
  - VBSA -Host -Variable, blocked, spanned, ASA print control characters
  - VBSM -Host -Variable, blocked, spanned, machine print control codes
  - VE -Host -Variable, ESDS
  - VEA -Host -Variable, ESDS, ASA print control characters
  - VEM -Host -Variable, ESDS, machine print control codes
  - VK -Host -Variable, KSDS
  - VKA -Host -Variable, KSDS, ASA print control characters
  - VKM -Host -Variable, KSDS, machine print control codes
  - VR -Host -Variable, RRDS
  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes
- NUMBER:RECLENGTH=num -Record length (data only, no print control and length fields)
- NUMBER:KEYDISP=OLD/NEW/DEL -Key disposition for index/key sequential data sets [OLD]
  - OLD -Key disposition OLD (keep the existing key)
  - NEW -Key disposition NEW (insert record number as key)
  - DEL -Key disposition DEL (delete key from record)
- NUMBER:KEYPOSITION=num -Key position (>=1) for index/key sequential data sets [1]
- NUMBER:KEYLENGTH=num -Key length (>=1) for index/key sequential data sets [8]
- STRING:LIKE='str' -Specifies the name of the model data set from which the attributes are to be copied
- SWITCH:RENEW -Enforce new allocation by a remove of an existing file

4.3.121 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- STRING:NAME='str' -Name of the subsystem
- STRING:PARM='str' -Parameter list for the subsystem (comma separated)

4.3.122 OBJECT *SPACE*

SYNOPSIS

```
HELP:   Platform dependent space definition parameter
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC
TYPE:   OBJECT
SYNTAX: SPACE(PRIMARY=num,SECONDARY=num,DIRBLOCKS=num,VOLSER='str',VOLUNIT='str',DATACLASS ↩
    ='str',MGMTCLASS='str',STORCLASS='str',NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG, ↩
    ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG,RELEASE=YES/NO,UNMOVABLE,CONTIG,PERMANENT,RLS= ↩
    NRI/CR/CRE)
```

DESCRIPTION The space definition object collects several parameters for platforms where data sets are preallocated like z/OS® and where clylinders or tracks must be predefined for a file allocation.

The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

All size values must be given in megabyte (MiB). The calculation in the platform dependent units (tracks, cylinders, ...) are done inernally. Note: 1 MiB = 1024 KiB = 1024*1024 Byte

ARGUMENTS

- NUMBER:PRIMARY=num -Primary size in megabytes
- NUMBER:SECONDARY=num -Secondary size in megabytes
- NUMBER:DIRBLOCKS=num -Directory blocks for partitioned data sets
- STRING:VOLSER='str' -Volume serial number
- STRING:VOLUNIT='str' -Volume unit
- STRING:DATACLASS='str' -Data class
- STRING:MGMTCLASS='str' -Management class
- STRING:STORCLASS='str' -Storage class
- NUMBER:NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for normal end [CATALOG]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for abnormal end [DELETE]
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete

  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:RELEASE=YES/NO -Release unused space when file is closed [SEQ=YES, PDS=NO]
  - YES -Yes, do it
  - NO -No, don't do it
- SWITCH:UNMOVABLE -Allocate space unmovable [FALSE]
- SWITCH:CONTIG -Allocate space contiguously [FALSE]
- SWITCH:PERMANENT -Allocate space permanently [FALSE]
- NUMBER:RLS=NRI/CR/CRE -Specifies the type of record level sharing (RLS) being done f or a specific data set [NONE]
  - NRI -No Read Integrity
  - CR -Consistent Read
  - CRE -Consistent Read Explicit

4.3.123 OBJECT *FL4*

SYNOPSIS

```
HELP:   Write a FLAMFILE record-oriented
PATH:   XCNV.OUTPUT.SAV.FILE.FIO
TYPE:   OBJECT
SYNTAX: FL4(NAME/FLAMFILE='str',MEMBER='str',REMAIN,APPEND,CMPMOD=CX7/CX8/VR8/ADC/NDC, ↩
    ENCMOD=OFF/AES/FLAMENC,BLKMOD=UNBLOCKED/BLOCKED,MAXBUF=num,MAXREC=num,FLFORG=SEQ/EDS/KDS ↩
    /RDS/LDS,FLFRCF=VAR/FIX/UND/TXT/DLM/VB/FB/V2B-XBE/V4B-XBE/V4A-ICL/V4E-ICL/V2B-ILE/V2B- ↩
    IBE/V4B-IBE,FLFRCL=num,FLFRCD=NONE/LF-ASCII/CR-ASCII/NL-ASCII/CRLF-ASCII/LF-EBCDIC/CR- ↩
    EBCDIC/NL-EBCDIC/CRLF-EBCDIC,FLFBLS=num,FLFCDP=REWIND/UNLOAD/LEAVE,FLFCOD=SYSTEM/ASCII/ ↩
    EBCDIC,SPLMOD=SERIAL/PARALLEL,SPLNBR=num,SPLSIZ=num,ORGORG=SEQ/EDS/KDS/RDS/LDS,ORGRCF= ↩
    VAR/FIX/UND/TXT/DLM/VB/FB/V2B-XBE/V4B-XBE/V4A-ICL/V4E-ICL/V2B-ILE/V2B-IBE/V4B-IBE,ORGRCL ↩
    =num,ORGBLS=num,ORGPRN=NONE/ASA/MCC,ORGRCD=NONE/LF-ASCII/CR-ASCII/NL-ASCII/CRLF-ASCII/LF ↩
    -EBCDIC/CR-EBCDIC/NL-EBCDIC/CRLF-EBCDIC,ORGSYS=DOS/WIN/UNIX/MAC/ZOS/VSE/ZVM/AIX/ZLINUX/ ↩
    AS400/OS2/HPUX/MPE/SUNOS/SOLARIS/BS2000/SINIX/VMS/NONSTOP/UNISYS/OFS/DUX/AMIGA,DEVICE= ↩
    FILE/USER,NONHDR,NONFIL,COMENT='str',PASSWD='bin'/DEFAULT,KMELIB='str',KMEFUC='str', ↩
    KMEPAR='str',EXTCMP='str',EXTDCO='str',ACSTIM=num,MODTIM=num,BLKORI,PRISPC=num,SECSPC= ↩
    num,VOLSER='str',VOLUNT='str',DATCLS='str',MGMCLS='str',STOCLS='str',DSPSTA=NEW/SHR/MOD/ ↩
    OLD,DSPNRM=CATALOG/DELETE/KEEP/UNCATALOG,DSPABO=CATALOG/DELETE/KEEP/UNCATALOG,KEYPOS=num ↩
    ,KEYLEN=num,SECINF=YES/NO/IGNORE/MEMBER,NOPATH)
```

DESCRIPTION This object defines the FLAM4 write operation for files. This method writes each data list entry as a record to a FLAMFILE of version 4 or older. It uses the FLAM4 record interface to write members to a FLAMFILE. The parameters below are explained in detail by the FLAM command of FLCL.

ARGUMENTS

- STRING:NAME/FLAMFILE='str' -Name/URL of FLAMFILE to write
- STRING:MEMBER='str' -Rename the member in the FLAMFILE
- SWITCH:REMAIN -Remain existing files (don't overwrite) [FALSE]
- SWITCH:APPEND -Append the member to this FLAMFILE (not useful with encryption) [FALSE]
- NUMBER:CMPMOD=CX7/CX8/VR8/ADC/NDC -FLAM Mode [ADC]
  - CX7 -Compression encoding as printable characters

  - CX8 -Compression encoding as binary data
  - VR8 -Enhanced compression encoding as binary data
  - ADC -Advanced Data Compression
  - NDC -No data compression
- NUMBER:ENCMOD=OFF/AES/FLAMENC -Encryption Mode [NONE]
  - OFF -Disable encryption
  - AES -AES encryption
  - FLAMENC -FLAM encryption
- NUMBER:BLKMOD=UNBLOCKED/BLOCKED -Block mode [unblocked]
  - UNBLOCKED -Each segment starts on new record (VSAM KSDS only)
  - BLOCKED -Segment can start within a record (recommended)
- NUMBER:MAXBUF=num -Maximal buffer size for FLAM matrix [32768]
- NUMBER:MAXREC=num -Maximal amount of records per matrix [255]
- NUMBER:FLFORG=SEQ/EDS/KDS/RDS/LDS -Organization of the FLAMFILE [SEQ]
  - SEQ -Sequential access [DEFAULT]
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)
  - LDS -Linear data set (VSAM-LDS)
- NUMBER:FLFRCF=VAR/FIX/UND/TXT/DLM/VB/FB/V2B-XBE/V4B-XBE/V4A-ICL/V4E-ICL/V2B-ILE/V2B-IBE/V4B-IBE -Record format for the FLAMFILE [FIX]
  - VAR -Variable
  - FIX -Fix
  - UND -Undefined
  - TXT -Text/Stream
  - DLM -Stream with special delimiter
  - VB -Variable blocked
  - FB -Fix blocked
  - V2B-XBE -Variable 2 byte length field without the length in big endian
  - V4B-XBE -Variable 4 byte length field without the length in big endian
  - V4A-ICL -Variable 4 byte length field in ASCII inclusive the length
  - V4E-ICL -Variable 4 byte length field in EBCDIC inclusive the length
  - V2B-ILE -Variable 2 byte length field inclusive the length in little endian
  - V2B-IBE -Variable 2 byte length field inclusive the length in big endian
  - V4B-IBE -Variable 4 byte length field inclusive the length in big endian
- NUMBER:FLFRCL=num -Record length for the FLAMFILE [512]
- NUMBER:FLFRCD=NONE/LF-ASCII/CR-ASCII/NL-ASCII/CRLF-ASCII/LF-EBCDIC/CR-EBCDIC/NL-EBCDIC/CRLF-EBCDIC -Record delimiter for the FLAMFILE (only for CX7)
  - NONE -No record separation
  - LF-ASCII -Record separation with line feed in ASCII (0x0A)
  - CR-ASCII -Record separation with carriage return in ASCII (0x0D)

  - NL-ASCII -Record separation with new line in ASCII (0x85)
  - CRLF-ASCII -Record separation with carriage return line feed in ASCII (0x0D0A)
  - LF-EBCDIC -Record separation with line feed in EBCDIC (0x25)
  - CR-EBCDIC -Record separation with carriage return in EBCDIC (0x0D)
  - NL-EBCDIC -Record separation with new line in EBCDIC (0x15)
  - CRLF-EBCDIC -Record separation with carriage return line feed in EBCDIC (0x0D25)
- NUMBER:FLFBLS=num -Block size for the FLAMFILE (mainly host related)
- NUMBER:FLFCDP=REWIND/UNLOAD/LEAVE -Close disposition for the FLAMFILE (mainly host r elated)
  - REWIND -Close disposition:Rewind (default)
  - UNLOAD -Close disposition:Unload
  - LEAVE -Close disposition:Leave
- NUMBER:FLFCOD=SYSTEM/ASCII/EBCDIC -Character set for string encoding in the FLAMFILE [SYSTEM]
  - SYSTEM -System specific encoding
  - ASCII -String encoding in ASCII
  - EBCDIC -String encoding in EBCDIC
- NUMBER:SPLMOD=SERIAL/PARALLEL -Split mode [NONE] (only for z/OS)
  - SERIAL -Serial splitting (after 1-4096 Megabyte)
  - PARALLEL -Parallel splitting (in 2, 3 or 4 files)
- NUMBER:SPLNBR=num -Split number (2-4 files for parallel splitting (only for z/OS))
- NUMBER:SPLSIZ=num -Split size (1-4096 MB for serial splitting (only for z/OS))
- NUMBER:ORGORG=SEQ/EDS/KDS/RDS/LDS -Organization of the original file [from origin]
  - SEQ -Sequential access [DEFAULT]
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)
  - LDS -Linear data set (VSAM-LDS)
- NUMBER:ORGRCF=VAR/FIX/UND/TXT/DLM/VB/FB/V2B-XBE/V4B-XBE/V4A-ICL/V4E-ICL/V2B-ILE/V2B-IBE/V4B-IBE -Record format for the original file [from origin]
  - VAR -Variable
  - FIX -Fix
  - UND -Undefined
  - TXT -Text/Stream
  - DLM -Stream with special delimiter
  - VB -Variable blocked
  - FB -Fix blocked
  - V2B-XBE -Variable 2 byte length field without the length in big endian
  - V4B-XBE -Variable 4 byte length field without the length in big endian
  - V4A-ICL -Variable 4 byte length field in ASCII inclusive the length
  - V4E-ICL -Variable 4 byte length field in EBCDIC inclusive the length

  - V2B-ILE -Variable 2 byte length field inclusive the length in little endian
  - V2B-IBE -Variable 2 byte length field inclusive the length in big endian
  - V4B-IBE -Variable 4 byte length field inclusive the length in big endian
- NUMBER:ORGRCL=num -Record length for the original file [from origin]
- NUMBER:ORGBLS=num -Block size for the original file [from origin]
- NUMBER:ORGRCD=NONE/LF-ASCII/CR-ASCII/NL-ASCII/CRLF-ASCII/LF-EBCDIC/CR-EBCDIC/NL-EBCDIC/CRLF-EBCDIC -Record delimiter for the original file [from origin]
  - NONE -No record separation
  - LF-ASCII -Record separation with line feed in ASCII (0x0A)
  - CR-ASCII -Record separation with carriage return in ASCII (0x0D)
  - NL-ASCII -Record separation with new line in ASCII (0x85)
  - CRLF-ASCII -Record separation with carriage return line feed in ASCII (0x0D0A)
  - LF-EBCDIC -Record separation with line feed in EBCDIC (0x25)
  - CR-EBCDIC -Record separation with carriage return in EBCDIC (0x0D)
  - NL-EBCDIC -Record separation with new line in EBCDIC (0x15)
  - CRLF-EBCDIC -Record separation with carriage return line feed in EBCDIC (0x0D25)
- STRING:ORGSYS=DOS/WIN/UNIX/MAC/ZOS/VSE/ZVM/AIX/ZLINUX/AS400/OS2/HPUX/MPE/SUNOS/SOLARIS/BS2000/SINIX/VMS/NONSTOP/UNISYS/OFS/DUX/AMIGA -Operating system for the original file [from origin]
  - DOS -MS DOS
  - WIN -MS Windows
  - UNIX -UNIX (general)
  - MAC -Apple MAC OS
  - ZOS -IBM z/OS
  - VSE -IBM z/VSE
  - ZVM -IBM z/VM
  - AIX -IBM AIX (Power)
  - ZLINUX -IBM z/Linux
  - AS400 -IBM AS400 /iSeries
  - OS2 -IBM OS/2
  - HPUX -HP-UX
  - MPE -HP MPE
  - SUNOS -SUN OS
  - SOLARIS -SUN /Open Solaris
  - BS2000 -Siemens BS2000 OSD
  - SINIX -Siemens Nixdorf Unix
  - VMS -HP Open VMS /DEC-VMS
  - NONSTOP -HP Non Stop /Tandem
  - UNISYS -UNISYS
  - OFS -DEC OFS1
  - DUX -DEC UNIX
  - AMIGA -Amiga /Motorola

- NUMBER:DEVICE=FILE/USER -Device used to store the FLAMFILE [FLATFILE]
  - FILE -Write to a file
  - USER -Use user IO
- SWITCH:NONHDR -Switch to write no file header [FALSE]
- SWITCH:NONFIL -Switch to write no file name in file header [FALSE]
- STRING:COMENT='str' -Comment for the member in the FLAMFILE [internal use]
- STRING:PASSWD='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['']
  - DEFAULT -FLAM4 default password
- STRING:KMELIB='str' -Library name for FKME ['']
- STRING:KMEFUC='str' -Function name of FKME ['']
- STRING:KMEPAR='str' -Parameter for the call of FKME ['']
- STRING:EXTCMP='str' -Name of user exit for compression
- STRING:EXTDCO='str' -Name of user exit for decompression
- NUMBER:ACSTIM=num -Define the last access time for this file
- NUMBER:MODTIM=num -Define the last modification time for this file
- SWITCH:BLKORI -Enforce block-oriented write of records (32752) with FLAM4
- NUMBER:PRISPC=num -Primary size in megabytes
- NUMBER:SECSPC=num -Secondary size in megabytes
- STRING:VOLSER='str' -Volume serial number
- STRING:VOLUNT='str' -Volume unit
- STRING:DATCLS='str' -Data class
- STRING:MGMCLS='str' -Management class
- STRING:STOCLS='str' -Storage class
- NUMBER:DSPSTA=NEW/SHR/MOD/OLD -Disposition status
  - NEW -Disposition status NEW (brand new file)
  - SHR -Disposition status SHR (read exiting file)
  - MOD -Disposition status MOD (append/modify file)
  - OLD -Disposition status OLD (overwrite existing file)
- NUMBER:DSPNRM=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for normal end
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep
  - UNCATALOG -Close disposition uncatalog
- NUMBER:DSPABO=CATALOG/DELETE/KEEP/UNCATALOG -Close disposition for abnormal end
  - CATALOG -Close disposition catalog
  - DELETE -Close disposition delete
  - KEEP -Close disposition keep

  - UNCATALOG -Close disposition uncatalog

- NUMBER:KEYPOS=num -Key position (>=1) for index/key sequential data sets [1]
- NUMBER:KEYLEN=num -Key length (>=1) for index/key sequential data sets [8]
- NUMBER:SECINF=YES/NO/IGNORE/MEMBER -Define handling of secure information
  - YES -Create secure information at compression
  - NO -Do not store any additional secure information
  - IGNORE -Ignore any security violations on decompression
  - MEMBER -Only member-specific security informations are verified
- SWITCH:NOPATH -Exclude directory path of the original file in the compressed data [FALSE]

4.3.124 PARAMETER *ORGPRN*

SYNOPSIS

```
HELP:   Print control byte for ASA or machine control character [from origin]
PATH:   XCNV.OUTPUT.SAV.FILE.FIO.FL4
TYPE:   NUMBER
SYNTAX: ORGPRN=NONE/ASA/MCC
```

DESCRIPTION Print control character handling is a special feature for record oriented files of mainframe system. For print control, the ASA or machine control charaters are encoded in the first byte of the record.

If an attribute with such a control character is known on write operation, you can decide whether it will be part of the output (RETAIN), written to the output if required (DETACH) or never written (ERASE).

SELECTIONS

- NONE -No print control character
- ASA -ASA print control character
- MCC -Machine print control codes

4.3.125 OBJECT *ENV*

SYNOPSIS

```
HELP:   Environment
PATH:   XCNV.OUTPUT.SAV.FILE
TYPE:   OBJECT
SYNTAX: ENV[(KEYWORD='str',VALUE='str')...]
```

DESCRIPTION With the object ENV you can overrule up to 32 environment variables for the write operation. This is useful to define (for example) another LANG variable to emulate another platform for character conversion.

ARGUMENTS

- STRING:KEYWORD='str' -Keyword of environment variable
- STRING:VALUE='str' -Value of environment variable

4.3.126 OBJECT *DIR*

SYNOPSIS

```
HELP:   Defines how to handle directories and various file types if wildcards used
PATH:   XCNV
TYPE:   OBJECT
SYNTAX: DIR(LINK,ALIAS,HIDDEN,RECURSIVE,ARCHIVE,ONEFILESYSTEM,TAPE)
```

DESCRIPTION This object defines multiple switches that control how input files are found. This is especially important when using wildcards in input filenames.

There are switches to enable recursion into subdirectories, following symbolic links, finding hidden files, recursion into archive files, include files on tapes, resolving aliases and enforcing to not leave the filesystem.

If the recursive flag is set, the filename portion of the input file string is searched for recursively. If a directory path containing wildcards is present, the filename portion is searched for recursively in all paths that match the directory path pattern, e.g. "**/path/**.txt" matches my/path/hugo.txt, another/path/hugo.txt as well as my/path/subdir/hugo.txt. Without the recursive flag, the last file would not be found.

If the archive switch is enabled, FLAMFILES, TARBALLS, ZIP files and other kinds of supported archives are searched. All the matching members are extracted. In other words, archives are interpreted like subdirectories if the flag is enabled.

If the tape switch is enabled, then datasets stored on tapes are searched. If the name of such a file matches the pattern, then it might occur that console input and a manual intervention is required to retrieve the respective dataset(s).

If the link switch is enabled, symbolic links are followed (otherwise they are ignored). Links are mainly supported on modern file systems. For dataset aliases on mainframe systems, see the alias flag.

With the alias switch enabled, datasets names which are aliases of the real dataset are resolved to their actual names (otherwise they are ignored). Dataset aliases only exist on mainframe systems. This flag has no effect on other platforms.

If a transparent read method (e.g. read.auto()) is used, then the archive flag is enabled by default and cannot be disabled.

ARGUMENTS

- `SWITCH:LINK -Include symbolic links`
- `SWITCH:ALIAS -Resolve data set aliases`
- `SWITCH:HIDDEN -Include hidden files`
- `SWITCH:RECURSIVE -Include sub directories`
- `SWITCH:ARCHIVE -Include archives (FLAMFILEs, ZIP, ...)`
- `SWITCH:ONEFILESYSTEM -Don't cross file system boundaries`
- `SWITCH:TAPE -Include files deposited on tapes`

4.3.127 OBJECT *LOG*

SYNOPSIS

```
HELP:   Logging Parameter
PATH:   XCNV
TYPE:   OBJECT
SYNTAX: LOG(MESSAGE(),DESTINATION.{})
```

DESCRIPTION The LOG parameter allows flexible adjustment for the logging of messages. It is possible to specify the destination as well as which kind of message will be written.

For more information please have a look at the man pages for each parameter of LOG.

4.3.128 OBJECT *MESSAGE*

SYNOPSIS

```
HELP:   Select type of messages to log
PATH:   XCNV.LOG
TYPE:   OBJECT
SYNTAX: MESSAGE(NONE,ERROR,ERRTRACE,WARNING,NOTICE,DEBUG,ABOUT,VERSION,INPUT,PARAMETER, ↩
    STATISTIC,INFO,LICID,LICENSE,PROGRESS,ALL,DEFAULT,MINIMAL,ERRONLY)
```

DESCRIPTION The log message specification is a classification of the messages to write to the selected log destination. Each type is selectable individually by adding its argument inside this object.

For added convenience, the special types NON, ALL, MINIMAL select the proper message types with just one argument.

If no message specification is given, a DEFAULT is used which writes out messages of these types:

- ERROR
- ERRTRACE
- WARNING
- NOTICE
- PARAMETER
- STATISTIC
- INFO
- LICID

ARGUMENTS

- `SWITCH:NONE -No messages will be logged`
- `SWITCH:ERROR -Error messages will be logged`
- `SWITCH:ERRTRACE -Error trace will be logged`
- `SWITCH:WARNING -Warnings will be logged`
- `SWITCH:NOTICE -Notices will be logged`
- `SWITCH:DEBUG -Debug information will be logged`
- `SWITCH:ABOUT -About information will be logged`
- `SWITCH:VERSION -Version information will be logged`
- `SWITCH:INPUT -Input data will be logged (dangerous for passwords)`
- `SWITCH:PARAMETER -Parsed parameters will be logged`
- `SWITCH:STATISTIC -Statistics will be logged`
- `SWITCH:INFO -Requested information will be logged`
- `SWITCH:LICID -License ID will be logged`
- `SWITCH:LICENSE -License summary will be logged`
- `SWITCH:PROGRESS -Progress bar will be logged`
- `SWITCH:ALL -All messages are logged`
- `SWITCH:DEFAULT -Relevant messages are logged`
- `SWITCH:MINIMAL -Only important messages are logged`
- `SWITCH:ERRONLY -Only errors and error trace are logged`

4.3.129 OVERLAY *DESTINATION*

SYNOPSIS

```
HELP:   Destination for log messages
PATH:   XCNV.LOG
TYPE:   OVERLAY
SYNTAX: DESTINATION.{STREAM()/FILE()/SYSTEM()}
```

DESCRIPTION The log destination parameter defines where the log messages will be written.

Each log message starts with the current time and date.

All the possible destinations allow to specify a custom identification string which will be printed after time and date of each message. This is done with the IDENT argument.

If no IDENT specified the default will be *owner.program.command*

If no log destination defined stream will be used as default.

4.3.130 OBJECT *STREAM*

SYNOPSIS

```
HELP:   Log to a stream
PATH:   XCNV.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: STREAM(FORMAT=STANDARD/DIALOG,IDENT='str',STDOUT)
```

DESCRIPTION The STREAM object allows the log messages to be written to the standard output or the standard error output. Standard error output is used if STDOUT is not specified.

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- SWITCH:STDOUT -Log messages to STDOUT instead of STDERR

4.3.131 OBJECT *FILE*

SYNOPSIS

```
HELP:   Log to a flat file
PATH:   XCNV.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: FILE(FORMAT=STANDARD/DIALOG,IDENT='str',NAME='str')
```

DESCRIPTION The flat file option allows the log messages to be written to a normal file. The name of the file must be specified for this destination. The log messages will be appended to this file if it already exists.

For the log file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   NAME='~.MYLOG(LOG1)'                  ; PDS on z/OS
   NAME='<SYSUID>.MYLOG02'               ; PS dataset on z/OS
   NAME='~/mylog3.txt'                   ; Unix path name
   NAME='<HOME>\logs\mylog4.txt'         ; Windows path name
```

ATTENTION: Parallel process cannot share the same log file

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- STRING:NAME='str' -Filename for appending log messages

4.3.132 OBJECT *SYSTEM*

SYNOPSIS

```
HELP:   Use system logging facility
PATH:   XCNV.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: SYSTEM(IDENT='str')
```

DESCRIPTION The SYSTEM destination allows the log messages to be written to the logging system of the operating system.

On Unix and z/OS® systems the syslog() call is used. On Windows® the event logging facility is used.

```
NOTE: On Windows(R) this requires the registry key entry:
      HKEY_LOCAL_MACHINE
        SYSTEM
          CurrentControlSet
            Services
              EventLog
                Application
                  FLM
```

This registry key will be created by using the DLL flevent.dll

ARGUMENTS

- STRING:IDENT='str' -Identifier used to identify log entries

4.4 COMMAND *ICNV*

SYNOPSIS

```
HELP:   Only character conversion
PATH:   LIM.flcl
TYPE:   OBJECT
SYNTAX: :> flcl ICNV(IN='str'/STREAM/DUMMY,OUT='str'/STREAM/DUMMY,FALLOC(),RECORD,METHOD= ↩
    AUTO/BLOCK/RECORD/KEEP/SKIP,CASE=UPPER/LOWER/FOLD/SUPPER/SLOWER/USRTAB,FROM='str'/ ↩
    DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,MODE=STOP/IGNORE/SUBSTITUTE,BOM,TO='str'/DEFAULT/ ↩
    ASCII/EBCDIC,ENL2LF,ELF2NL,SUBCHR[num/SYSTEM...],SYSTAB=ICONV,USRTAB='str'/NPAS/SEPA/ ↩
    DELA/DLAX,REPORT='str'/STDOUT/STDERR,REMAIN,APPEND,REMOVE,DIR(),LOG())
```

DESCRIPTION The ICNV command converts files from one character encoding to another encoding set. More precisely, it converts the input file from the encoding given for the FROM option to an output file where the encoding is given in the TO option. Both encodings must be defined and supported by the platform you are running on.

During this it supports:

- Record or block oriented conversion
- Stop, ignore or substitution of invalid or incomplete character
- A free defined substitution character list
- Different system substitution / transliteration tables
- A user defined substitution table to change the system tables
- Case management (upper, lower, folding and special casing)
- Transliteration between subsets (mapping) and normalization
- BOM management and automatic detection of char sets
- Replacement of EBCDIC new line (0x15) by line feed (0x25)
- Reporting of ignored or substituted characters

The ICNV command runs the character conversion as part of the read operation of the Frankenstein Limes Universal Converter (FLUC). As default the block oriented binary read and write of files is used. With the keyword RECORD it will be possible to switch to record oriented binary read and write IO operations. Only binary formatting of the data is used. All the other conversion capabilities of FLUC are not available with this command.

To get syntax information, please use:

```
flcl SYNTAX ICNV
```

To get help for a parameter, please use:

```
flcl HELP ICNV.parameter[.parameter[...]]
```

To read the manual page for a parameter, please use:

```
flcl MANPAGE ICNV.parameter[.parameter[...]]
```

To generate the user manual for the command, please use:

```
flcl GENDOCU ICNV=filename
```

Parameters can be defined by the command line (directly or per file) or by properties taken from the corresponding property file

EXAMPLES

```
flcl ICNV in='input.txt' out='output.txt'
          from='UTF-8' to='ISO-8859-1' mode=ignore
```

converts input from unicode to the old West-European encoding ISO-8859-1 and discards non convertable characters in the output

```
flcl ICNV in='input.txt' out='output.txt'
          from='ISO-8859-1' to='UTF-8'
```

converts input from the old West-European encoding ISO-8859-1 to unicode.

ARGUMENTS

- `SWITCH:REMAIN -Remain existing files (don't overwrite) [FALSE]`
- `SWITCH:REMOVE -Remove original file after successful processing [FALSE]`

4.4.1 PARAMETER *IN*

SYNOPSIS

```
HELP:   Input file/URL [required]
PATH:   ICNV
TYPE:   STRING
SYNTAX: IN='str'/STREAM/DUMMY
```

DESCRIPTION Defines the input file which is read. The encoding for the input file must be defined with the FROM option.

SELECTIONS

- STREAM -Read from stdin or write to stdout
- DUMMY -Read EOF or write nothing

4.4.2 PARAMETER *OUT*

SYNOPSIS

```
HELP:   Output file/URL [required]
PATH:   ICNV
TYPE:   STRING
SYNTAX: OUT='str'/STREAM/DUMMY
```

DESCRIPTION Defines the output file which is written. The encoding for the output file is defined by the TO code.

SELECTIONS

- STREAM -Read from stdin or write to stdout
- DUMMY -Read EOF or write nothing

4.4.3 OBJECT *FALLOC*

SYNOPSIS

```
HELP:   Platform dependent file allocation parameter for output
PATH:   ICNV
TYPE:   OBJECT
SYNTAX: FALLOC(ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZE=num,RECFORMAT=NONE/BIN/TXT/ ↩
    DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX- ↩
    REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR ↩
    /FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/ ↩
    VRA/VRM,RECLENGTH=num,KEYDISP=OLD/NEW/DEL,KEYPOSITION=num,KEYLENGTH=num,SUBSYSTEM(), ↩
    SPACE(),LIKE='str',RENEW)
```

DESCRIPTION The file allocation object collects several parameters for platforms which are record oriented based on a catalog system like z/OS® and where the space, volume and other things must be predefined. The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

But also on distributed platforms like (WIN, UNIX, MAC, ...) FLAM supports record oriented file types to keep the record information on such platforms. For this case the different record format for the open world can be defined. To read such a special file format on an open platform the correct record format must also be defined. On a catalog based system, this is not required. If the record format in the catalog are not the same as specified an error will occur.

ARGUMENTS

- NUMBER:ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [SEQ]
  - SEQ -Sequential data set [DEFAULT]
  - LIB -Library (ZOS-PDSE)
  - PDS -Partitioned data set (ZOS-PDS)
  - USS -UNIX files (path based)
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)
- NUMBER:BLKSIZE=num -Block size [system]
- NUMBER:RECFORMAT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format [auto]
  - NONE -No format defined
  - BIN -Open -Binary (undefined) data (only the last record are shorter then record length)
  - TXT -Open -Text data with standard delimiter (result is a stripped variable length records)
  - DLM -Open -Records separated by a special delimiter with the result of variable length records
  - VAR -Open -Variable length record (if necessary define length format)
  - VAR-ASA -Open -Variable length record with ASA print control character (if necessary define length format)
  - VAR-MCC -Open -Variable length record with machine print control codes (if necessary define length format)
  - FIX -Open -Fix length record (file size =N * record length)
  - FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
  - FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
  - VAR-REL -Open -Variable relative record (if necessary define length format)
  - VAR-REL-ASA -Open -Variable relative record with ASA print control character (if necessary define length format)
  - VAR-REL-MCC -Open -Variable relative record with machine print control codes (if necessary define length format)
  - FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
  - FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size = N * (record length + 9))
  - FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size = N * (record length + 9))
  - F -Host -Fixed-length, unblocked
  - FA -Host -Fixed-length, unblocked, ASA print control characters
  - FM -Host -Fixed-length, unblocked, machine print control codes
  - FS -Host -Fixed-length, standard
  - FSA -Host -Fixed-length, standard, ASA print control characters
  - FSM -Host -Fixed-length, standard, machine print control codes

  - FB -Host -Fixed-length, blocked
  - FBA -Host -Fixed-length, blocked, ASA print control characters
  - FBM -Host -Fixed-length, blocked, machine print control codes
  - FBS -Host -Fixed-length, blocked, standard
  - FBSA -Host -Fixed-length, blocked, standard, ASA print control characters
  - FBSM -Host -Fixed-length, blocked, standard, machine print control codes
  - FE -Host -Fixed-length, ESDS
  - FEA -Host -Fixed-length, ESDS, ASA print control characters
  - FEM -Host -Fixed-length, ESDS, machine print control codes
  - FK -Host -Fixed-length, KSDS
  - FKA -Host -Fixed-length, KSDS, ASA print control characters
  - FKM -Host -Fixed-length, KSDS, machine print control codes
  - FR -Host -Fixed-length, RRDS
  - FRA -Host -Fixed-length, RRDS, ASA print control characters
  - FRM -Host -Fixed-length, RRDS, machine print control codes
  - U -Host -Undefined-length
  - UA -Host -Undefined-length, ASA print control characters
  - UM -Host -Undefined-length, machine print control codes
  - V -Host -Variable
  - VA -Host -Variable, ASA print control characters
  - VM -Host -Variable, machine print control codes
  - VS -Host -Variable, spanned
  - VSA -Host -Variable, spanned, ASA print control characters
  - VSM -Host -Variable, spanned, machine print control codes
  - VB -Host -Variable, blocked
  - VBA -Host -Variable, blocked, ASA print control characters
  - VBM -Host -Variable, blocked, machine print control codes
  - VBS -Host -Variable, blocked, spanned
  - VBSA -Host -Variable, blocked, spanned, ASA print control characters
  - VBSM -Host -Variable, blocked, spanned, machine print control codes
  - VE -Host -Variable, ESDS
  - VEA -Host -Variable, ESDS, ASA print control characters
  - VEM -Host -Variable, ESDS, machine print control codes
  - VK -Host -Variable, KSDS
  - VKA -Host -Variable, KSDS, ASA print control characters
  - VKM -Host -Variable, KSDS, machine print control codes
  - VR -Host -Variable, RRDS
  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes

- NUMBER:RECLENGTH=num -Record length (data only, no print control and length fields)
- NUMBER:KEYDISP=OLD/NEW/DEL -Key disposition for index/key sequential data sets [OLD]
  - OLD -Key disposition OLD (keep the existing key)
  - NEW -Key disposition NEW (insert record number as key)

  - DEL -Key disposition DEL (delete key from record)

- NUMBER:KEYPOSITION=num -Key position (>=1) for index/key sequential data sets [1]
- NUMBER:KEYLENGTH=num -Key length (>=1) for index/key sequential data sets [8]
- STRING:LIKE='str' -Specifies the name of the model data set from which the attributes are to be copied
- SWITCH:RENEW -Enforce new allocation by a remove of an existing file

4.4.4 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   ICNV.FALLOC
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- STRING:NAME='str' -Name of the subsystem
- STRING:PARM='str' -Parameter list for the subsystem (comma separated)

4.4.5 OBJECT *SPACE*

SYNOPSIS

```
HELP:   Platform dependent space definition parameter
PATH:   ICNV.FALLOC
TYPE:   OBJECT
SYNTAX: SPACE(PRIMARY=num,SECONDARY=num,DIRBLOCKS=num,VOLSER='str',VOLUNIT='str',DATACLASS ↩
    ='str',MGMTCLASS='str',STORCLASS='str',NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG, ↩
    ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG,RELEASE=YES/NO,UNMOVABLE,CONTIG,PERMANENT,RLS= ↩
    NRI/CR/CRE)
```

DESCRIPTION The space definition object collects several parameters for platforms where data sets are preallocated like z/OS® and where clylinders or tracks must be predefined for a file allocation.

The supported parameters are platform dependent. Only parameters relevant for a certain platform are used. This way its possible to store files on special volumes, units, and with SMS-classes without any JCL usage on z/OS®.

All size values must be given in megabyte (MiB). The calculation in the platform dependent units (tracks, cylinders, ...) are done inernally. Note: 1 MiB = 1024 KiB = 1024*1024 Byte

ARGUMENTS

- NUMBER:PRIMARY=num –Primary size in megabytes
- NUMBER:SECONDARY=num –Secondary size in megabytes
- NUMBER:DIRBLOCKS=num –Directory blocks for partitioned data sets
- STRING:VOLSER='str' –Volume serial number
- STRING:VOLUNIT='str' –Volume unit
- STRING:DATACLASS='str' –Data class
- STRING:MGMTCLASS='str' –Management class
- STRING:STORCLASS='str' –Storage class
- NUMBER:NORMALDISP=CATALOG/DELETE/KEEP/UNCATALOG –Close disposition for normal end [CATALOG]
  - CATALOG –Close disposition catalog
  - DELETE –Close disposition delete
  - KEEP –Close disposition keep
  - UNCATALOG –Close disposition uncatalog
- NUMBER:ABORTDISP=CATALOG/DELETE/KEEP/UNCATALOG –Close disposition for abnormal end [DELETE]
  - CATALOG –Close disposition catalog
  - DELETE –Close disposition delete
  - KEEP –Close disposition keep
  - UNCATALOG –Close disposition uncatalog
- NUMBER:RELEASE=YES/NO –Release unused space when file is closed [SEQ=YES, PDS=NO]
  - YES –Yes, do it
  - NO –No, don't do it
- SWITCH:UNMOVABLE –Allocate space unmovable [FALSE]
- SWITCH:CONTIG –Allocate space contiguously [FALSE]
- SWITCH:PERMANENT –Allocate space permanently [FALSE]
- NUMBER:RLS=NRI/CR/CRE –Specifies the type of record level sharing (RLS) being done for a specific data set [NONE]
  - NRI –No Read Integrity
  - CR –Consistent Read
  - CRE –Consistent Read Explicit

4.4.6 PARAMETER *RECORD*

SYNOPSIS

```
HELP:   Use record-oriented file IO instead of block-oriented IO
PATH:   ICNV
TYPE:   SWITCH
SYNTAX: RECORD
```

DESCRIPTION Switches from block oriented file io to record oriented file io. This option is mainly relevant for mainframes.

4.4.7 PARAMETER *METHOD*

SYNOPSIS

```
HELP:   Enforces record or block-oriented conversion [AUTO]
PATH:   ICNV
TYPE:   NUMBER
SYNTAX: METHOD=AUTO/BLOCK/RECORD/KEEP/SKIP
```

DESCRIPTION The method of character conversion defines how the input data fields are handled. A data field can be one block or a number of records dependent from how the data is read and converted before character conversion is done. Is the data field a list of records then the record oriented method shall be used. Is the data field a single block the block oriented conversion shall be used. This means that the METHOD must only be set to enforce record or block orientation against the usual way of processing the data.

- Block Orientation - This means all records are handled as one block and the record length information will go lost. If incomplete characters occur at the end of this block it will be remembered as rest and copied to the next block. Using this method there will be no interruption of the data stream.
- Record Orientation - This method ensures that each record contains only complete character sequences. An incomplete character sequence will only be accepted at the end of the last record and handled also as rest for the record in the next data field. To enforce record orientation processing of one block of data there will be no difference to block orientation conversion, besides for processing the last block.

Thus it makes only sense to switch for a record organized data field to block orientation conversion. But this is very critical, as the length information will go lost. This means that there will be no chance to retrieve the record information, besides the record contains some delimiters which can be used to build a new list of record lengths. So we recommend to use the METHOD switch only if you know what you are doing. In some cases analysis of the error trace might result in the correct method that can be used to convert the text.

If an incomplete byte sequence progresses in the next record, then it can be helpful to enforce block oriented conversion, because the record length conversion can be invalid. On the other side, if the next record starts with a new valid byte sequence, then a skip of the damaged byte sequence in the last record can be helpful.

If SKIP is selected then each incomplete character at the end of a record is recorded in the report file with a special code point 0xFFFFFF including the really skipped byte sequence. If this problem happens in the last block you can enforce record oriented conversion with SKIP to convert the data and the incomplete rest of the last block. They are written to the report file. Normally the corresponding MODE is used for the final block to prevent a rest.

Please don't mix up SKIP as METHOD for the rest of a record contained in a data field and the IGNORE as MODE for the content of a record as unit of work.

The default method depends on the data. If the data is marked as block-oriented or the record type is text with delimiters, then the method BLOCK is used. In all other cases, record orientation with the method KEEP is used.

SELECTIONS

- `AUTO -Automatic orientation (block or record with keep)`

- BLOCK -Block-oriented conversion (all data is one chunk)
- RECORD -Record-oriented conversion (check record boundaries)
- KEEP -Keep (Move a rest from the end to begin of next record)
- SKIP -Skip a rest at the end of a record (not recommended)

4.4.8 PARAMETER *CASE*

SYNOPSIS

```
HELP:   Define upper, lower or special case conversion [NONE]
PATH:   ICNV
TYPE:   NUMBER
SYNTAX: CASE=UPPER/LOWER/FOLD/SUPPER/SLOWER/USRTAB
```

DESCRIPTION With the case modifier, you can perform conversion to upper or lower case as well as apply case folding and some special case conversions.

Case folding converts all characters to lower case but is more compatible with language specific rules for doing so. For example, this may be useful for case-insensitive string comparison if the strings contain non-Latin characters (code point < 128).

This mapping is done for each code point in front of substitution or transliteration. The respective case is accomplished on the basis of a separate Unicode conversion table in memory. This conversion can be manipulated if user mapping is activated and mapping definitions (*) are defined through the provided user table.

The options UPPER and LOWER convert characters without expansion, i.e. returned strings will never grow in length. Note, however, that case folding and the special options (convert e.g. ß to SS) can result in an expansion of data.

All these conversions are pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- UPPER -Change character to upper case
- LOWER -Change character to lower case
- FOLD -Change character to folded case (for compare)
- SUPPER -Change character to special upper case
- SLOWER -Change character to special lower case
- USRTAB -Change character according to user table

4.4.9 PARAMETER *FROM*

SYNOPSIS

```
HELP:   Conversion from this CCSID [auto detection]
PATH:   ICNV
TYPE:   STRING
SYNTAX: FROM='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)

- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (auto-detect)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`
- `BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)`
- `BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)`

4.4.10 PARAMETER *MODE*

SYNOPSIS

```
HELP:   Define handling of non-convertible characters
PATH:   ICNV
TYPE:   NUMBER
SYNTAX: MODE=STOP/IGNORE/SUBSTITUTE
```

DESCRIPTION With the MODE selection you can define how an invalid or incomplete character is handled inside of a unit by this component. All possible modes are listed at the end of this chapter.

This MODE is valid for the content of each unit (block or record). At the end of a unit there can be an incomplete character. Such a rest will be handled dependent on the METHOD you choose. The rest of the last unit of a data field will be concatenated to the first unit of the next data field. For all other records in a data field the handling of such a rest will be defined by the respective METHOD.

The SKIP case for METHOD is only relevant for incomplete byte sequences at the end of a unit. The ignore and substitution mode is only effective for all characters in the middle of the unit, besides an incomplete byte sequence occurs in the last unit. Please don't mix up the IGNORE case for MODE with the SKIP case of METHOD.

If a system or user substitution table is used the stop, ignore or substitute are only relevant for characters not managed by the CCSIDs and the substitution tables. This means that the USRTAB and SYSTAB are added to the capabilities of character conversion and only for the rest the chosen MODE is used.

4.4.10.1 CONSTANT *STOP*

SYNOPSIS

```
HELP:   Stop at the first non-convertible character
PATH:   ICNV.MODE
TYPE:   NUMBER
SYNTAX: STOP
```

DESCRIPTION Stops conversion at the first incomplete or invalid character which can not be converted. An error message is given with the position of abortion and the value in the input stream. An entry will be (STP()) written to the report file to log this event. To be compatible with other conversion tools, this is the default case, but we recommend to use IGNORE or SUBSTITUTE to prevent any abortion.

indexterm:[Constant STOP}

4.4.10.2 CONSTANT *IGNORE*

SYNOPSIS

```
HELP:   Ignore non-convertible characters
PATH:   ICNV.MODE
TYPE:   NUMBER
SYNTAX: IGNORE
```

DESCRIPTION Ignores incomplete or invalid characters and writes a report entry for each character. IGNORE is a substitution by no corresponding character that means no substitution table is required.

indexterm:[Constant IGNORE}

4.4.10.3 CONSTANT *SUBSTITUTE*

SYNOPSIS

```
HELP:   Substitute non-convertible characters
PATH:   ICNV.MODE
TYPE:   NUMBER
SYNTAX: SUBSTITUTE
```

DESCRIPTION Substitutes incomplete or invalid character and writes a report entry for each character. For SUBSTITUTE you can define a substitution character list (SUBCHR). If no SUBCHR is provided then this option substitutes invalid or incomplete characters with code point 0x00001A. If a SUBCHR is defined all invalid or incomplete characters will be substituted by the defined list of code points that can be up to 8. If you specify a SUBCHR the substitution mode will be enabled.

indexterm:[Constant SUBSTITUTE}

4.4.11 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Write byte order mark in front of the output text
PATH:   ICNV
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the write byte order mark keyword is defined, then the corresponding BOM sign is written at the start of the output stream (block oriented) or at the beginning of each record. If this BOM keyword is not used, then the BOM sign will not be written (this is valid independent of the chosen encoding/CCSID). If a valid BOM sign is located in the middle of the input stream, it is removed from the data. If you define the BOM keyword and the target charset contains only single bytes or no valid BOM sign, then no byte order mark will be written to the output file.

If an invalid BOM sign is recognized, then it will be ignored and causes no entry in the report file. If the byte order changes in the middle of text and the BOM handling switch is not activated, the conversion will stop and an error message is thrown.

By default, the write BOM keyword is set to OFF.

4.4.12 PARAMETER *TO*

SYNOPSIS

```
HELP:   Conversion to this CCSID [UTF-8]
PATH:   ICNV
TYPE:   STRING
SYNTAX: TO='str'/DEFAULT/ASCII/EBCDIC
```

DESCRIPTION The TO code defines the encoding string or CCSID for the output text stream. The default value of this CCSID for read operations is UTF-8. For write operations, the default value is derived from the environment variable LANG. If LANG is not set, the default ASCII or EBCDIC system-codepage is used, depending on the platform.

To get a list of supported encoding strings, please use the command below:

```
   flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs please use:

```
   flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
   CCSID='37'    will be equal to CCSID='IBM037'
   CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit. Optionally, you can use one of the keywords below to use the system-specific default ASCII or EBCDIC CCSID.

SELECTIONS

- `DEFAULT –Use default CCSID (UTF-8)`
- `ASCII –Use default ASCII CCSID (environment)`
- `EBCDIC –Use default EBCDIC CCSID (environment)`

4.4.13 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   ICNV
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.4.14 PARAMETER *ELF2NL*

SYNOPSIS

```
HELP:   Replace EBCDIC line feed (0x25) by new line (0x15) on output
PATH:   ICNV
TYPE:   SWITCH
SYNTAX: ELF2NL
```

DESCRIPTION The replacement of EBCDIC line feed control characters (0x25) by new line characters (0x15) at conversion to EBCDIC can be activated on non EBCDIC (ASCII) systems or deactivated on EBCDIC systems by enabling this switch. This is for example required to prepare text files on open platforms for USS on z/OS. The usual delimiter for text in ASCII and UTF is 0x0A and will normally be translated to 0x25 in EBCDIC. On z/OS(USS), however, 0x15 (NL) is expected.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

To ensure a correctly delimited text record, only a line feed and not a carriage return (0x0D) line feed (0x0A) is used. If text formatting is used with method=NL, this will result in a single LF on distributed systems and the internal conversion will be activated by default to produce a valid text file for z/OS under USS on Windows, Unix and other open platforms.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.4.15 PARAMETER *SUBCHR*

SYNOPSIS

```
HELP:   Default substitution character list [0x1A] (maximal 8 code points)
PATH:   ICNV
TYPE:   NUMBER
SYNTAX: SUBCHR[num/SYSTEM...]
```

DESCRIPTION As substitution character (SUBCHR) you can define a list with up to 8 UNICODE points which are used for the substitution of invalid or incomplete characters if the corresponding code point is not defined in the substitution table (see SUBTAB). If you assign to it the keyword SYSTEM the value 0x1A is written as default substitution character.

If you activate substitution and no SUBCHR is defined, then unsupported characters will be replaced by CODEPOINT 0x00001A.

On the other hand SUBCHR will replace every undefined code point in the substitution table, what is the same behavior when using a USRTAB with corresponding SUBCHR values for every possible code point (000000-200000). If SUBCHR but no MODE is defined the substitution mode will be activated.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `SYSTEM -Use system specific substitution char (0x1A for ASCII/UTF, 0x3F for EBCDIC)`

4.4.16 PARAMETER *SYSTAB*

SYNOPSIS

```
HELP:   Define a preloaded system substitution table
PATH:   ICNV
TYPE:   NUMBER
SYNTAX: SYSTAB=ICONV
```

DESCRIPTION With the system table (SYSTAB) you can load a substitution table with a predefined transliteration schema. Entries in the USRTAB can be used to override predefined transliteration entries or add missing transliteration code points. The options listed at the end of this chapter are available.

If a substitution table is used then the MODE option (STOP/IGNORE/SUBSTITUTE) is only relevant for characters that cannot be converted or substituted.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `ICONV -Preload with ICONV//TRANSLIT substitution table`

4.4.17 PARAMETER *USRTAB*

SYNOPSIS

```
HELP:   Name of the file containing the user conversion table
PATH:   ICNV
TYPE:   STRING
SYNTAX: USRTAB='str'/NPAS/SEPA/DELA/DLAX
```

DESCRIPTION With this parameter, you can choose a predefined subset or standard manipulation for the system tables via keyword (see `flcl help "xcnv.inp .sav.fil.cnv.chr.usrtab"` for available keywords). Alternatively, you can use a file containing a user defined substitution table (USRTAB).

For the user table file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
USRTAB=NPAS                          ; use XOEV predefined subset
USRTAB=DELA                          ; German Latin single byte subset
USRTAB='~.MYUSRTAB(USRTAB1)'         ; PDS on z/OS
USRTAB='<SYSUID>.MYUSRT02'           ; PS dataset on z/OS
USRTAB='~/myusrtab3.txt'             ; Unix path name
USRTAB='<HOME>\usrtab\myusrtab4.txt' ; Windows path name
```

A user defined substitution table has the syntax listed below:

```
usr_tab_file         -> usr_definition_list
usr_definition_list  -> usr_definition usr_definition_list
                     |  EMPTY
usr_definition       -> '(' codepoint_definition ')'
codepoint_definition -> cp_activation
                     |  cp_deactivation
                     |  cp_transliteration
                     |  cp_standard_mapping
                     |  cp_usr_case_mapping
cp_activation        -> ['+'] CODEPOINT  [ '-' CODEPOINT ]
cp_deactivation      ->  '-'  CODEPOINT  [ '-' CODEPOINT ]
cp_transliteration   -> ['+'] CODEPOINT '=' code_point_list
cp_standard_mapping  ->  '*'  CODEPOINT '=' code_point_list
cp_usr_case_mapping  ->  '^'  CODEPOINT '=' code_point_list
code_point_list      -> CODEPOINT separator code_point_list
                     |  CODEPOINT
                     |  EMPTY
separator            -> '/'
                     |  ','
                     |  ';'
```

Note: In the description below "/" is used as code point separator.

The optional signs in front of a code point (CP) have the following meaning:

```
'+' - a valid transliteration (only if CP not in the target set)
'*' - a valid mapping         (this translation is always done)
'^' - a case mapping/folding  (only if case=usrtab is activated)
'-' - an invalid definition   (removes CP for subset definitions)
```

If no sign is used, then a transliteration for a valid code point is defined. Valid code points are accepted, invalid code points result in in an error. Error handling depends on the mode (STOP,IGNORE, SUBSTITUTE). Definition of + code points without code point list, thus without = is equal definition of valid code points. With an asterisk $*$, you can define an enforced mapping for this code point. For example, this can be used to delete this character if no code point list is provided or to translate this character always in another value. This can also be used to convert code points outside of a subset into this subset.

In contrast to the transliteration (+ is only done if a character doesn't exist in the target encoding), the mapping is always done for each character. Invalid (−) code points can be used to deactivate a code point. To activate (+) or deactivate (−) a range of code points you can use the minus mark (−) followed by one sub code point. The range goes from the smaller code point to the bigger one. For example:

```
(-00-7F) deactivate all US-ASCII code points
(+39-30) activate all decimal digits
```

The range operator (−) with the optional plus sign (+) can also be used to activate code points.

To define an own case mapping or folding (required for certain subsets) the caret sign (^) can be used. This mapping is only done if the user table is activated for case mapping (CASE=USRTAB).

To define transliterations/substitutions/mappings an assignment (=) of a code point list is required. The code point list can contain a maximum of 8 code points. If no code point list is specified, then the character is ignored. A transliteration to itself simply activates this code point. The same is valid for mapping definitions. A code point is a hexadecimal number representing a 21 bit UNICODE point.

```
  00000000 to 001FFFFF hex
```

If a code point is not defined in the substitution table (USRTAB and/or SYSTAB), the appropriate substitution character (SUBCHR) is used. If SUBCHR is not set, no substitution is performed. Depending on the MODE, character conversion stops with an error (STOP) or ignores the character (IGNORE).

Text before and after brackets are comments.

Between "(" and "=" or "/" or ")" you can define hex digits until the first non-hex digit. All non-hex digits up to the next separator are interpreted as comment. Leading whitespace is ignored.

```
  REPLACE GERMAN SZ (00DF=0073/0073)                  WITH ss
THIS IS AN EXAMPLE FOR COMMENTS IN CODE POINT LIST:
  REPLACE EURO MARK (20AC= 45#E / 55#U / 52#R / 4F#O) WITH EURO
  REPLACE BOM MARK  (EFFF=)                           WITH NOTHING
```

Leading zeros are allowed. If you don't supply a hex value, then 0x00 is used.

```
  REPLACE GERMAN SZ (00DF=/)                   WITH 0x00 0x00
  REPLACE EURO MARK (20AC=00000045/00055/000052/000004F) WITH EURO
```

ATTENTION: Please don't use parentheses "()" or the operators in your comments.

To describe your own subsets, you can use a user table without a system table. The USRTAB can also be used to overwrite or add transliterations when a system substitution table (for example SYSTAB=ICONV) is used. The transliteration works recursively, that is if one of the substitution code points is not in the target set, this substitution will be used instead and so on.

```
  REPLACE GERMAN OE (0000D6=004F/0045)       WITH O E
  REPLACE GERMAN SZ (0000DF=0073/0073)       WITH s s
  REPLACE EURO MARK (0020AC=00D6/00DF/52/4F) WITH OE SZ R O
```

If you have a EURO sign in your text and convert it to *Latin1*, the resulting byte string will be *D6DF524F*. On the other hand, if you convert it to *ASCII*, the byte string will be *4F457373524F*.

The mapping as described above is always done and is also done recursive including the transliteration result. For example, if you define a mapping `(*30=39)` then the target data won't have any zero in the resulting text.

By replacing other code points recursively, you could easily cause infinite loops. To prevent this, the amount of replacements and the length of one replacement is limited to a maximum of 64. Note that this could still result in a large expansion of data. Therefore, be careful about defining recursive substitutions.

A sample user table for the ICONV system table to change the transliteration of German umlauts to AE, OE or UE is located in the SAMPLE directory under CCUTDEXL. Another sample user table called CCUTNPAS defines the *string.latin* subset (XOEF) which is mainly used for statutory reporting.

The order of definition is the order of processing the definitions. For exampel, if you define a mapping or a transliteration for code point X and later you make this code point invalid, then the code point is invalid. On the other hand, you can first deactivate all code points, then activate your subset and define your transliterations (best fit) or mappings.

For example: To define a non-expansion best fit mapping a user table can be used to delete all combined characters for single byte code pages. For the ICONV system transliteration table, a sample user table can be found under the name CCUTBFM1.

Some other subset user table definitions were added. For example:

- currently valid SEPA characters (<128) for money transfers (CCUTSEPA)
- characters valid in ISO-8859-15, CP1252 and IBM1142 (CCUTDELA)
- characters valid in ISO-8859-15, CP1252, IBM1142 and XOEF (CCUTDLAX)

All user table definitions are pre-calculated once at the beginning of execution to reduce the required CPU time for the conversion. So, the use of a user table increases the cost to open a file but has no effect on CPU utilization when converting the data.

To minimize the effort and to simplify the usage of user table, the most common subsets can be selected by keyword. In this case, a pre-calculated load module (DLL) is used. The subset definition can be for UTF (21 Bit UNICODE) or UCS (16 Bit UNICODE). If it is a UCS subset, then no valid code point is greater than 0xFFFF and you must use UCS CCSIDS to ensure that no code point greater than 0xFFFF is accepted.

If you have a working user table definition for a popular subset or system table manipulation, you can request a keyword for it. Please report this requirement over our issue tracking system

```
http://www.flam.de/en/technology/support/issues/
```

and attach the corresponding user table definition.

SELECTIONS

- `NPAS -UCS subset for statutory reporting (New passport, XOEF, String.Latin)`
- `SEPA -UCS subset of valid SEPA character smaller then 128`
- `DELA -UCS subset for German Latin of IBM1141, ISO8859-15 and CP1252`
- `DLAX -UCS subset for German Latin of IBM1141, ISO8859-15, CP1252 and XOEF`

4.4.18 PARAMETER *REPORT*

SYNOPSIS

```
HELP:   Name of the report file to log substitutions
PATH:   ICNV
TYPE:   STRING
SYNTAX: REPORT='str'/STDOUT/STDERR
```

DESCRIPTION With the report capability, you can log every substitution (stop / ignore / substitute / translit / skip / ...) to a log file. For the report file, all rules and replacements are valid, which are described under "FILENAME HANDLING" above. For example:

```
REPORT=STDERR                           ; is SYSPRINT on z/OS
REPORT='~.MYREPORT(REPORT1)'            ; PDS on z/OS
REPORT='<SYSUID>.MYREPRT2'              ; PS dataset on z/OS
REPORT='~/myreport3.txt'                ; Unix path name
REPORT='<HOME>\reports\myreport4.txt'   ; Windows path name
```

A substitution report has the syntax listed below:

```
report_file         -> sub_definition_list
sub_definition_list -> sub_definition sub_definition_list
                    |  EMPTY
sub_definition      -> SUB '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    -> XLT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string '='
                           code_point_list byte_string ')'
                    |  IGN '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  STP '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  IDT '(' UNIT ':' OFFSET '+' POSITION ':'
                           code_point byte_string ')'
                    |  SKP '(' UNIT ':' OFFSET '+' POSITION ':'
```

```
                          code_point byte_string ')'
                    |  UKW '(' UNIT ':' OFFSET '+' POSITION ':'
                          code_point byte_string ')'
  code_point_list   -> CODEPOINT '/' code_point_list
                    |  EMPTY
  code_point        -> CODEPOINT
                    |  EMPTY
  byte_string       -> '(' BYTESTRING ')'
                    |  EMPTY
```

A comment behind brackets explains the reason for substitution:

```
'invalid character'
'incomplete character'
'incomplete character at the end of a record'
```

The notations STP (stop), IGN (ignore), SUB (substitute), XLT (transliteration), IDT (identical) or SKP (skip) before round brackets indicate the action taken. If the task can not be determined then UKW (unknown) is used. A ignore of an invalid character is realized as substitution with nothing. All substitutions with a length of 0 are indicated with IGN (ignore). The position of such an abnormal/irregular character is coded inside the braces:

- UNIT number of records or blocks (depends on the data)
- OFFSET byte offset up to this unit in the complete data stream
- POSITION number of bytes inside the unit where the activity was done

All counting starts with 0. The first block or record has unit 0.

The position is relative to the data stream which was presented to the character conversion module. After conversions and formatting the position values can be far off from the input file which was read originally. For example, when reading a text file with text formatting to records, then delimiting characters are not included in the byte offset, the units are records and the position indicates which character was the issue for this log entry.

The CODEPOINTs (UNICODE) are taken from the defined substitution table The BYTESTRINGs are taken from the input (read characters) and output data (written characters). BYTESTRINGs and CODEPOINTs are provided if they are known in the cases of such an irregularity. In some cases, these values can be missing.

For interpreting the report file, you must know that only the values inside round brackets are meaningful. Text before or behind round brackets are comments, but this may be subject to changes.

SELECTIONS

- `STDOUT -Write output to stdout`
- `STDERR -Write output to stderr`

4.4.19 PARAMETER *APPEND*

SYNOPSIS

```
HELP:   Append data to output file [FALSE]
PATH:   ICNV
TYPE:   SWITCH
SYNTAX: APPEND
```

DESCRIPTION With APPEND the data is written at the end of the file. If the file does not exist then it will be created. If the file exists then the data will be appended to the file and not overwritten.

4.4.20 OBJECT *DIR*

SYNOPSIS

```
HELP:   Defines how to handle directories and various file types if wildcards used
PATH:   ICNV
TYPE:   OBJECT
SYNTAX: DIR(LINK,ALIAS,HIDDEN,RECURSIVE,ARCHIVE,ONEFILESYSTEM,TAPE)
```

DESCRIPTION This object defines multiple switches that control how input files are found. This is especially important when using wildcards in input filenames.

There are switches to enable recursion into subdirectories, following symbolic links, finding hidden files, recursion into archive files, include files on tapes, resolving aliases and enforcing to not leave the filesystem.

If the recursive flag is set, the filename portion of the input file string is searched for recursively. If a directory path containing wildcards is present, the filename portion is searched for recursively in all paths that match the directory path pattern, e.g. "**/path/**.txt" matches my/path/hugo.txt, another/path/hugo.txt as well as my/path/subdir/hugo.txt. Without the recursive flag, the last file would not be found.

If the archive switch is enabled, FLAMFILES, TARBALLS, ZIP files and other kinds of supported archives are searched. All the matching members are extracted. In other words, archives are interpreted like subdirectories if the flag is enabled.

If the tape switch is enabled, then datasets stored on tapes are searched. If the name of such a file matches the pattern, then it might occur that console input and a manual intervention is required to retrieve the respective dataset(s).

If the link switch is enabled, symbolic links are followed (otherwise they are ignored). Links are mainly supported on modern file systems. For dataset aliases on mainframe systems, see the alias flag.

With the alias switch enabled, datasets names which are aliases of the real dataset are resolved to their actual names (otherwise they are ignored). Dataset aliases only exist on mainframe systems. This flag has no effect on other platforms.

If a transparent read method (e.g. read.auto()) is used, then the archive flag is enabled by default and cannot be disabled.

ARGUMENTS

- `SWITCH:LINK -Include symbolic links`
- `SWITCH:ALIAS -Resolve data set aliases`
- `SWITCH:HIDDEN -Include hidden files`
- `SWITCH:RECURSIVE -Include sub directories`
- `SWITCH:ARCHIVE -Include archives (FLAMFILEs, ZIP, ...)`
- `SWITCH:ONEFILESYSTEM -Don't cross file system boundaries`
- `SWITCH:TAPE -Include files deposited on tapes`

4.4.21 OBJECT *LOG*

SYNOPSIS

```
HELP:   Logging Parameter
PATH:   ICNV
TYPE:   OBJECT
SYNTAX: LOG(MESSAGE(),DESTINATION.{})
```

DESCRIPTION The LOG parameter allows flexible adjustment for the logging of messages. It is possible to specify the destination as well as which kind of message will be written.

For more information please have a look at the man pages for each parameter of LOG.

4.4.22 OBJECT *MESSAGE*

SYNOPSIS

```
HELP:   Select type of messages to log
PATH:   ICNV.LOG
TYPE:   OBJECT
SYNTAX: MESSAGE(NONE,ERROR,ERRTRACE,WARNING,NOTICE,DEBUG,ABOUT,VERSION,INPUT,PARAMETER, ↩
    STATISTIC,INFO,LICID,LICENSE,PROGRESS,ALL,DEFAULT,MINIMAL,ERRONLY)
```

DESCRIPTION The log message specification is a classification of the messages to write to the selected log destination. Each type is selectable individually by adding its argument inside this object.

For added convenience, the special types NON, ALL, MINIMAL select the proper message types with just one argument.

If no message specification is given, a DEFAULT is used which writes out messages of these types:

- ERROR
- ERRTRACE
- WARNING
- NOTICE
- PARAMETER
- STATISTIC
- INFO
- LICID

ARGUMENTS

- `SWITCH:NONE -No messages will be logged`
- `SWITCH:ERROR -Error messages will be logged`
- `SWITCH:ERRTRACE -Error trace will be logged`
- `SWITCH:WARNING -Warnings will be logged`
- `SWITCH:NOTICE -Notices will be logged`
- `SWITCH:DEBUG -Debug information will be logged`
- `SWITCH:ABOUT -About information will be logged`
- `SWITCH:VERSION -Version information will be logged`
- `SWITCH:INPUT -Input data will be logged (dangerous for passwords)`
- `SWITCH:PARAMETER -Parsed parameters will be logged`
- `SWITCH:STATISTIC -Statistics will be logged`
- `SWITCH:INFO -Requested information will be logged`
- `SWITCH:LICID -License ID will be logged`
- `SWITCH:LICENSE -License summary will be logged`
- `SWITCH:PROGRESS -Progress bar will be logged`
- `SWITCH:ALL -All messages are logged`
- `SWITCH:DEFAULT -Relevant messages are logged`
- `SWITCH:MINIMAL -Only important messages are logged`
- `SWITCH:ERRONLY -Only errors and error trace are logged`

4.4.23 OVERLAY *DESTINATION*

SYNOPSIS

```
HELP:   Destination for log messages
PATH:   ICNV.LOG
TYPE:   OVERLAY
SYNTAX: DESTINATION.{STREAM()/FILE()/SYSTEM()}
```

DESCRIPTION The log destination parameter defines where the log messages will be written.

Each log message starts with the current time and date.

All the possible destinations allow to specify a custom identification string which will be printed after time and date of each message. This is done with the IDENT argument.

If no IDENT specified the default will be *owner.program.command*

If no log destination defined stream will be used as default.

4.4.24 OBJECT *STREAM*

SYNOPSIS

```
HELP:   Log to a stream
PATH:   ICNV.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: STREAM(FORMAT=STANDARD/DIALOG,IDENT='str',STDOUT)
```

DESCRIPTION The STREAM object allows the log messages to be written to the standard output or the standard error output. Standard error output is used if STDOUT is not specified.

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- SWITCH:STDOUT -Log messages to STDOUT instead of STDERR

4.4.25 OBJECT *FILE*

SYNOPSIS

```
HELP:   Log to a flat file
PATH:   ICNV.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: FILE(FORMAT=STANDARD/DIALOG,IDENT='str',NAME='str')
```

DESCRIPTION The flat file option allows the log messages to be written to a normal file. The name of the file must be specified for this destination. The log messages will be appended to this file if it already exists.

For the log file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   NAME='~.MYLOG(LOG1)'                  ; PDS on z/OS
   NAME='<SYSUID>.MYLOG02'               ; PS dataset on z/OS
   NAME='~/mylog3.txt'                   ; Unix path name
   NAME='<HOME>\logs\mylog4.txt'         ; Windows path name
```

ATTENTION: Parallel process cannot share the same log file

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- STRING:NAME='str' -Filename for appending log messages

4.4.26 OBJECT *SYSTEM*

SYNOPSIS

```
HELP:   Use system logging facility
PATH:   ICNV.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: SYSTEM(IDENT='str')
```

DESCRIPTION The SYSTEM destination allows the log messages to be written to the logging system of the operating system.

On Unix and z/OS® systems the syslog() call is used. On Windows® the event logging facility is used.

```
NOTE: On Windows(R) this requires the registry key entry:
      HKEY_LOCAL_MACHINE
        SYSTEM
          CurrentControlSet
            Services
              EventLog
                Application
                  FLM
```

This registry key will be created by using the DLL flevent.dll

ARGUMENTS

- STRING:IDENT='str' -Identifier used to identify log entries

4.5 COMMAND *XCHK*

SYNOPSIS

```
HELP:   Extended data validation
PATH:   LIM.flcl
TYPE:   OBJECT
SYNTAX: :> flcl XCHK(INPUT(),DIR(),LOG())
```

DESCRIPTION The XCHK command can be used, for example, to analyze files or members inside archives of various formats (without unpacking to a temporary file), generate their hash values, etc.

For example, you can generate or verify hash values before or after any conversion, i.e. after decoding, decryption, decompression or character conversion.

The XCHK command was designed to provide the full capacity of the powerful Frankenstein Limes Universal Converter (FLUC) without the write overhead. It supports all parameters of XCNV INPUT.

To get syntax information, please use:

```
flcl SYNTAX XCHK
```

To get help for the parameters, please use:

```
flcl HELP XCHK.parameter[.parameter[...]]
```

To read the manual page for a parameter, please use:

```
flcl MANPAGE XCHK.parameter[.parameter[...]]
   or
flcl HELP XCHK.parameter[.parameter[...]] MAN
```

To generate the user manual for the command, please use:

```
flcl GENDOCU XCHK=filename
```

Parameters can be defined via command line (directly or per file) or through properties read from the corresponding property file.

EXAMPLES Reads file "xzfile.xz" and calculates the checksum using the default hash algorithm.

```
flcl xchk "input(sav.file(fio.blk(name=xzfile.xz) cnv.hsh() ))"
```

Reads file "xzfile.xz", decompresses it and calculates the checksum using the default hash algorithm.

```
flcl xchk "input(sav.file(fio.blk(name=xzfile.xz) cnv.lxz()
                                                  cnv.hsh() ))"
```

Reads file "xzfile.xz", calculate the checksum using the SHA-1 hash algorithm, decompresses the file and calculates the checksum of the decompressed data using the MD5 hash algorithm.

```
flcl xchk "input(sav.file(fio.blk(name=xzfile.xz) cnv.hsh(algo=SHA1)
                                                  cnv.lxz()
                                                  cnv.hsh(algo=MD5)))"
```

A simplified command for hash calculation and verification is available under the name *HASH*.

4.5.1 OBJECT *INPUT*

SYNOPSIS

```
HELP:   Inbound parameter
PATH:   XCHK
TYPE:   OBJECT
SYNTAX: INPUT(NET.{},SAV.{})
```

DESCRIPTION Over the object INPUT you can define how the data is read.

4.5.2 OVERLAY *NET*

SYNOPSIS

```
HELP:   Network [NONE]
PATH:   XCHK.INPUT
TYPE:   OVERLAY
SYNTAX: NET.{IPN()/IPS()/MQS()}
```

DESCRIPTION With the overlay NET you can choose the protocol for the remote read or write access. Below You can find the currently supported protocols and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.

4.5.3 OBJECT *IPN*

SYNOPSIS

```
HELP:   IP-Native
PATH:   XCHK.INPUT.NET
TYPE:   OBJECT
SYNTAX: IPN(HOST='str',PORT='str',USER='str',AUTH='str')
```

DESCRIPTION The IP native (IPN) method supports IPv4 and IPv6 network communication for remote access to files, archives and other sources and targets of the FLAM infrastructure. For this client-server architecture, the FLIES server must be started and running in the IPN mode listening on a dedicated port (17996) on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or DNS name) and the corresponding service port (number or name) must be specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES server is required to handle requests.

ARGUMENTS

- `STRING:HOST='str' -IP address [localhost]`
- `STRING:PORT='str' -Port number [17996]`
- `STRING:USER='str' -User name [optional]`
- `STRING:AUTH='str' -Authentication data [optional]`

4.5.4 OBJECT *IPS*

SYNOPSIS

```
HELP:   IP-Secure (SSL/TLS)
PATH:   XCHK.INPUT.NET
TYPE:   OBJECT
SYNTAX: IPS(HOST='str',PORT='str',USER='str',AUTH='str')
```

DESCRIPTION The IP secure (IPS) method supports IPv4 and IPv6 network communication with SSL/TLS for remote access to files, archives and other sources and targets of the FLAM infrastructure. For this client-server architecture, the FLIES server must be started and running in the IPS mode listening on a dedicated port (17996) on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or DNS name) and the corresponding service port (number or name) must be specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES server is required to handle requests.

ARGUMENTS

- `STRING:HOST='str' -IP address [localhost]`
- `STRING:PORT='str' -Port number [17996]`
- `STRING:USER='str' -User name [optional]`
- `STRING:AUTH='str' -Authentication data [optional]`

4.5.5 OBJECT *MQS*

SYNOPSIS

```
HELP:   MQ-Series
PATH:   XCHK.INPUT.NET
TYPE:   OBJECT
SYNTAX: MQS(QMANAGER='str',QUEUE='str',USER='str',AUTH='str')
```

DESCRIPTION The MQ-Series (MQS) method supports IBM® message queuing for remote access to files, archives and other sources and targets of the FLAM infrastructure. For this client-server architecture, the FLIES server must be started and running in MQS mode listening on a dedicated queue on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES server is required to handle requests.

ARGUMENTS

- `STRING:QMANAGER='str' -Queue manager name [""]`
- `STRING:QUEUE='str' -Queue name [""]`
- `STRING:USER='str' -User name [optional]`
- `STRING:AUTH='str' -Authentication data [optional]`

4.5.6 OVERLAY *SAV*

SYNOPSIS

```
HELP:   Storage [FILE]
PATH:   XCHK.INPUT
TYPE:   OVERLAY
SYNTAX: SAV.{FILE()}
```

DESCRIPTION With the overlay SAV you can choose the kind of source or target for the read or write operation. Below You can find the currently supported procedures and the corresponding parameter explained in a dedicated chapter.

4.5.7 OBJECT *FILE*

SYNOPSIS

```
HELP:   File handling
PATH:   XCHK.INPUT.SAV
TYPE:   OBJECT
SYNTAX: FILE(FIO.{},CNV[{}...],FMT.{},ENV[()...])
```

DESCRIPTION Over the object FILE you can define how a list of neutral FLAM5 elements with type, length, value and attributes is read from a file. For this you can choose a specific kind of I/O (block, record, text, FLAM4, ...), up to 32 conversions (decoding, decryption, decompression, character set, ...) and a formatting method (binary, text, xml, ...) for the data.

4.5.8 OVERLAY *FIO*

SYNOPSIS

```
HELP:   File IO [BLK]
PATH:   XCHK.INPUT.SAV.FILE
TYPE:   OVERLAY
SYNTAX: FIO.{BLK()/REC()/TXT()/FL4()}
```

DESCRIPTION With the overlay FIO you can choose different kind of file IO methods for read operations.

4.5.9 OBJECT *BLK*

SYNOPSIS

```
HELP:   Read a file block-oriented (mainly for open world)
PATH:   XCHK.INPUT.SAV.FILE.FIO
TYPE:   OBJECT
SYNTAX: BLK(NAME='str'/STREAM/DUMMY,BLKSIZ=num,DATTYP=BINARY/CHAR/TEXT/XML,PRNCTR=DETACH/ ↩
    RETAIN/ERASE,FRCBLK,ADDDLM,SUBSYS(),REMOVE)
```

DESCRIPTION This object defines the block oriented read operation for files. This method reads block size bytes as one chunk and gives this block simply in binary form to the subsequent conversion layer. This is the most famous and simplest way to read a file. This read method is useful for all non record oriented file formats.

You can also read record oriented data sets. In this case the block size defines how many records are processed in one chunk. If the next complete record too big for this block, it is copied as first record of the next block.

If you read records with FIOBLK there are no capability to store any attributes except the record length. Means print control character and slot number are lost. Record oriented data sets you can use FIOREC to manage record attributes separately. Print control character can remain in the record oriented data block or detached, but the attribute is lost after the read operation.

You can also enforce block orientation on a record-oriented device, in which case record boundaries are lost and the block is filled byte-wise. This will result in better performance, but a transparent read of different file formats and normal host data sets is not possible anymore. Additionally, you can also add the system delimiter behind each record to produce a valid block for text formatting.

ARGUMENTS

- STRING:NAME='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- NUMBER:DATTYP=BINARY/CHAR/TEXT/XML -Definition of data type for better handling [NONE]
  - BINARY -Binary data (not printable)
  - CHAR -Character data (no delimiter)
  - TEXT -Text data (with delimiter)
  - XML -XML data (like CHAR)
- SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]
- SWITCH:ADDDLM -Add a delimiter behind each record to produce a text block [FALSE]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.5.10 PARAMETER *BLKSIZ*

SYNOPSIS

```
HELP:   Block size for read operations [AUTO]
PATH:   XCHK.INPUT.SAV.FILE.FIO.BLK
TYPE:   NUMBER
SYNTAX: BLKSIZ=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.5.11 PARAMETER *PRNCTR*

SYNOPSIS

```
HELP:   Print control character handling if read ASA or MCC records [DETACH]
PATH:   XCHK.INPUT.SAV.FILE.FIO.BLK
TYPE:   NUMBER
SYNTAX: PRNCTR=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.5.12 OBJECT *SUBSYS*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   XCHK.INPUT.SAV.FILE.FIO.BLK
TYPE:   OBJECT
SYNTAX: SUBSYS(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
SUBSYS(NAME='name' parm='keyword,keyword=value,value')
SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.5.13 OBJECT *REC*

SYNOPSIS

```
HELP:   Read a file record-oriented (mainly for mainframe world)
PATH:   XCHK.INPUT.SAV.FILE.FIO
TYPE:   OBJECT
SYNTAX: REC(NAME='str'/STREAM/DUMMY,RECMOD=STOP/CUT/WRAP,PRNCTR=DETACH/RETAIN/ERASE,SUPPAD, ↩
    PADCHR='bin'/BINARY-ZERO/ASCII-BLANK/EBCDIC-BLANK/UTF08-BLANK/UTF16BE-BLANK/UTF16LE- ↩
    BLANK/UTF32BE-BLANK/UTF32LE-BLANK,FILORG=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZ=num,RECFMT= ↩
    NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC ↩
    /FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/ ↩
    FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/ ↩
    VKA/VKM/VR/VRA/VRM,RECLEN=num,RECCNT=num,LENFMT.{},CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/ ↩
    EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE,RECDLM='bin'/CR-ASCII ↩
    /LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF-EBCDIC/NL-EBCDIC/CRLF-EBCDIC/CR-UTF08/LF- ↩
    UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE/NL-UTF16BE/CRLF-UTF16BE/CR-UTF16LE/LF- ↩
    UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF-UTF32BE/NL-UTF32BE/CRLF-UTF32BE/CR-UTF32LE ↩
    /LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE,KEYPOS=num,KEYLEN=num,SUBSYS(),REMOVE)
```

DESCRIPTION This object defines the record-oriented read operation for files. This method reads records from a data set. You can define the record length and the behavior (STOP/CUT/WRAP) if the record is longer than the defined record length. You can activate the suppression of padding characters and set the padding character. You can define the file organization, block size, record format and a lot of other parameters (see below).

The record length is always the real data length after all deductions. For example, for a FBA with a LRECL of 121, the record length of FLAM is 120 because the print control character is normally an attribute and not a part of the record.

On record oriented systems (mainly mainframes), a lot of these parameters are known from the catalog and need not be defined. On other platforms (Windows, Unix, . . .), catalogs do not exist. On such systems, you must specify how the record-oriented file was written (record format, length format).

FLAM supports some record formats on mainframes which are not supported by the corresponding catalog system. For example, record formats with print control characters for VSAM data sets are supported by FLAM. If you know your data in a KSDS has a print control character in the first byte, you can choose the corresponding record format (VKA/VKM) to DETACH the print control character as attribute from the record.

The record I/O component also supports text files with delimiters. The record format *text* can be used to parse records form a delimiter-based text file, for which you can define the character set. It is also possible to parse binary data into records based on

a custom delimiter. For this, the record format DLM can be used. The record format BIN simply wraps the data into records of the provided record length. The final record may be shorter. With the FIX record format, the last record also has the provided record length. For variable length record formats (VAR) on non-record-oriented systems, the length format must be defined. All these formats are supported to transfer data between record-oriented and block-oriented systems.

For USS on z/OS the default length format for variable records conforms to SVC99 allocation with FILEDATA=RECORD. The record format TXT works like FILEDATA=TEXT if default settings used, aso.

Attributes (length, slot number of RRDS, print control character) are stored in front of each record on non-record-oriented systems. Example: If you read a RRDS with print control characters (FRA) and you write this on UNIX as record format VRA to a file, the file format looks like this:

```
LLLLnnnnnnnnCdddd..dddd
LLLLnnnnnnnnCdddd..dddd
LLLLnnnnnnnnCdddd..dddd
...
...
...
LLLLnnnnnnnnCdddd..dddd
LLLLnnnnnnnnCdddd..dddd
```

Where LLLL is the record length, nn…nn is the slot number C is the print control character and dd..dd is the record data. To read such a file on a UNIX system, the record format and length format (if you write with defaults, you can read with defaults) must be defined correctly.

ARGUMENTS

- STRING:NAME='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- NUMBER:RECMOD=STOP/CUT/WRAP -Mode used to read records [STOP]
  - STOP -Stop if record too long
  - CUT -Cut if record too long
  - WRAP -Wrap if record too long
- SWITCH:SUPPAD -Suppress trailing padding characters (useful for fixed records)
- STRING:PADCHR='bin'/BINARY-ZERO/ASCII-BLANK/EBCDIC-BLANK/UTF08-BLANK/UTF16BE-BLANK/UTF16LE-BLANK/UTF32BE-BLANK/UTF32LE-BLANK -Define padding character for suppression [automatic]
  - BINARY-ZERO -Padding with 0x00
  - ASCII-BLANK -Padding with 0x20
  - EBCDIC-BLANK -Padding with 0x40
  - UTF08-BLANK -Padding with 0x20
  - UTF16BE-BLANK -Padding with 0x0020
  - UTF16LE-BLANK -Padding with 0x2000
  - UTF32BE-BLANK -Padding with 0x00000020
  - UTF32LE-BLANK -Padding with 0x20000000
- NUMBER:FILORG=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [SEQ]
  - SEQ -Sequential data set [DEFAULT]
  - LIB -Library (ZOS-PDSE)
  - PDS -Partitioned data set (ZOS-PDS)

  - USS -UNIX files (path based)
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)

- NUMBER:RECFMT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format for read operations [NONE]
  - NONE -No format defined
  - BIN -Open -Binary (undefined) data (only the last record are shorter then record length)
  - TXT -Open -Text data with standard delimiter (result is a stripped variable length records)
  - DLM -Open -Records separated by a special delimiter with the result of variable length records
  - VAR -Open -Variable length record (if necessary define length format)
  - VAR-ASA -Open -Variable length record with ASA print control character (if necessary define length format)
  - VAR-MCC -Open -Variable length record with machine print control codes (if necessary define length format)
  - FIX -Open -Fix length record (file size =N * record length)
  - FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
  - FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
  - VAR-REL -Open -Variable relative record (if necessary define length format)
  - VAR-REL-ASA -Open -Variable relative record with ASA print control character (if necessary define length format)
  - VAR-REL-MCC -Open -Variable relative record with machine print control codes (if necessary define length format)
  - FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
  - FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size = N * (record length + 9))
  - FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size = N * (record length + 9))
  - F -Host -Fixed-length, unblocked
  - FA -Host -Fixed-length, unblocked, ASA print control characters
  - FM -Host -Fixed-length, unblocked, machine print control codes
  - FS -Host -Fixed-length, standard
  - FSA -Host -Fixed-length, standard, ASA print control characters
  - FSM -Host -Fixed-length, standard, machine print control codes
  - FB -Host -Fixed-length, blocked
  - FBA -Host -Fixed-length, blocked, ASA print control characters
  - FBM -Host -Fixed-length, blocked, machine print control codes
  - FBS -Host -Fixed-length, blocked, standard
  - FBSA -Host -Fixed-length, blocked, standard, ASA print control characters

  - FBSM -Host -Fixed-length, blocked, standard, machine print control codes
  - FE -Host -Fixed-length, ESDS
  - FEA -Host -Fixed-length, ESDS, ASA print control characters
  - FEM -Host -Fixed-length, ESDS, machine print control codes
  - FK -Host -Fixed-length, KSDS
  - FKA -Host -Fixed-length, KSDS, ASA print control characters
  - FKM -Host -Fixed-length, KSDS, machine print control codes
  - FR -Host -Fixed-length, RRDS
  - FRA -Host -Fixed-length, RRDS, ASA print control characters
  - FRM -Host -Fixed-length, RRDS, machine print control codes
  - U -Host -Undefined-length
  - UA -Host -Undefined-length, ASA print control characters
  - UM -Host -Undefined-length, machine print control codes
  - V -Host -Variable
  - VA -Host -Variable, ASA print control characters
  - VM -Host -Variable, machine print control codes
  - VS -Host -Variable, spanned
  - VSA -Host -Variable, spanned, ASA print control characters
  - VSM -Host -Variable, spanned, machine print control codes
  - VB -Host -Variable, blocked
  - VBA -Host -Variable, blocked, ASA print control characters
  - VBM -Host -Variable, blocked, machine print control codes
  - VBS -Host -Variable, blocked, spanned
  - VBSA -Host -Variable, blocked, spanned, ASA print control characters
  - VBSM -Host -Variable, blocked, spanned, machine print control codes
  - VE -Host -Variable, ESDS
  - VEA -Host -Variable, ESDS, ASA print control characters
  - VEM -Host -Variable, ESDS, machine print control codes
  - VK -Host -Variable, KSDS
  - VKA -Host -Variable, KSDS, ASA print control characters
  - VKM -Host -Variable, KSDS, machine print control codes
  - VR -Host -Variable, RRDS
  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes

- NUMBER:RECLEN=num -Record length for read operations [512]
- NUMBER:RECCNT=num -Amount of records handled in one read operation [128]
- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [auto detection]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)

  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)

- STRING:RECDLM='bin'/CR-ASCII/LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF-EBCDIC/NL-EBCDIC/CRLF-EBCDIC/CR-UTF08/LF-UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE/NL-UTF16BE/CRLF-UTF16BE/CR-UTF16LE/LF-UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF-UTF32BE/NL-UTF32BE/CRLF-UTF32BE/CR-UTF32LE/LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE -Delimiter used to define end of record
  - CR-ASCII -Delimiter:carriage return in ASCII (0x0D)
  - LF-ASCII -Delimiter:line feed in ASCII (0x0A)
  - NL-ASCII -Delimiter:new line in ASCII (0x85)
  - CRLF-ASCII -Delimiter:carriage return line feed in ASCII (0x0D0A)
  - CR-EBCDIC -Delimiter:carriage return in EBCDIC (0x0D)
  - LF-EBCDIC -Delimiter:line feed in EBCDIC (0x25)
  - NL-EBCDIC -Delimiter:new line in EBCDIC (0x15)
  - CRLF-EBCDIC -Delimiter:carriage return line feed in EBCDIC (0x0D25)
  - CR-UTF08 -Delimiter:carriage return in UTF-8 (0x0D)
  - LF-UTF08 -Delimiter:line feed in UTF-8 (0x0A)
  - NL-UTF08 -Delimiter:new line in UTF-8 (0xC285)
  - CRLF-UTF08 -Delimiter:carriage return line feed in UTF-8 (0x0D0A)
  - CR-UTF16BE -Delimiter:carriage return in UTF-16BE (0x000D)
  - LF-UTF16BE -Delimiter:line feed in UTF-16BE (0x000A)
  - NL-UTF16BE -Delimiter:new line in UTF-16BE (0x0085)
  - CRLF-UTF16BE -Delimiter:carriage return line feed in UTF-16BE (0x000D000A)
  - CR-UTF16LE -Delimiter:carriage return in UTF-16LE (0x0D00)
  - LF-UTF16LE -Delimiter:line feed in UTF-16LE (0x0A00)
  - NL-UTF16LE -Delimiter:new line in UTF-16LE (0x8500)
  - CRLF-UTF16LE -Delimiter:carriage return line feed in UTF-16LE (0x0D000A00)
  - CR-UTF32BE -Delimiter:carriage return in UTF-32BE (0x0000000D)
  - LF-UTF32BE -Delimiter:line feed in UTF-32BE (0x0000000A)
  - NL-UTF32BE -Delimiter:new line in UTF-32BE (0x00000085)
  - CRLF-UTF32BE -Delimiter:carriage return line feed in UTF-32BE (0x0000000D0000000A)
  - CR-UTF32LE -Delimiter:carriage return in UTF-32LE (0x0D000000)
  - LF-UTF32LE -Delimiter:line feed in UTF-32LE (0x0A000000)
  - NL-UTF32LE -Delimiter:new line in UTF-32LE (0x85000000)
  - CRLF-UTF32LE -Delimiter:carriage return line feed in UTF-32LE (0x0D0000000A000000)

- NUMBER:KEYPOS=num -Key position (>=1) for index/key sequential data sets [0=NONE]
- NUMBER:KEYLEN=num -Key length (>=1) for index/key sequential data sets [if KEYPOS>0 then 8]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.5.14 PARAMETER *PRNCTR*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   XCHK.INPUT.SAV.FILE.FIO.REC
TYPE:   NUMBER
SYNTAX: PRNCTR=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.5.15 PARAMETER *BLKSIZ*

SYNOPSIS

```
HELP:   Block size for read operations [AUTO]
PATH:   XCHK.INPUT.SAV.FILE.FIO.REC
TYPE:   NUMBER
SYNTAX: BLKSIZ=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.5.16 OVERLAY *LENFMT*

SYNOPSIS

```
HELP:   Format of the length field for open platforms
PATH:   XCHK.INPUT.SAV.FILE.FIO.REC
TYPE:   OVERLAY
SYNTAX: LENFMT.{INTEGER()/STRING()/BCD()/HOST()}
```

DESCRIPTION The overlay can be used to define the format of the length filed on open platforms which don't support record formats by the data management system. For example: If you read records on a mainframe and you want to write the records as records (not as binary or text data) on WINDOWS or UNIX then the data will be prefixed with the record length. The format of this length field can be changed over this union.

This length specification will be only used for variable length records. The default mapping method between record formats will be map each host record format the corresponding open variable format.

If the length format not defined the length field will be written as 32 bit integer in platform dependent byte order. This behavior corresponds to the FILEDATA=RECORD parameter for USS files on z/OS. Means if you write a record oriented file to USS without any format changed over this overlay then you can read the record with another MVS application using FILEDATA=RECORD.

If you choose record format text or binary then this will correspond to FILEDATA=TEXT or FILEDATA=BINARY and the length format specification will be ignored. To enable record format mapping based on the data type you can set the environment variable below:

```
FL_RECORD_FORMAT_MAPPING=DATATYPE
```

If this environment variable set, then the length format is only used if you explicitly set a variable record format of the open systems.

Since version 5.1.8, FLAM can detect the 4 byte length fields (integer or host) inside a data stream. We recommend to use one of these formats to have more comfort when reading.

4.5.17 OBJECT *INTEGER*

SYNOPSIS

```
HELP:   Use integer format for the length field
PATH:   XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT
TYPE:   OBJECT
SYNTAX: INTEGER(ENDIAN=SYSTEM/BIG/LITTLE,WIDTH=U16/U32/U64,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in integer format. You can define the byte order, the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [SYSTEM]
  - SYSTEM -Byte order of the current system
  - BIG -Big endian byte order
  - LITTLE -Little endian byte order
- NUMBER:WIDTH=U16/U32/U64 -Bit width/rate [B32]
  - U16 -16 bit unsigned integer
  - U32 -32 bit unsigned integer
  - U64 -64 bit unsigned integer
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.5.18 OBJECT *STRING*

SYNOPSIS

```
HELP:   Use string format for the length field
PATH:   XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT
TYPE:   OBJECT
SYNTAX: STRING(CHRSET=SYSTEM/ASCII/EBCDIC,COUNT=C04/C05/C06,BASE=B10,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in string format (a character per digit). You can define the character set, the character count, the base and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:CHRSET=SYSTEM/ASCII/EBCDIC -Character set [SYSTEM]
  - SYSTEM -Use system code page
  - ASCII -Use ASCII (UTF-8) code page
  - EBCDIC -Use EBCDIC code page
- NUMBER:COUNT=C04/C05/C06 -Character count [C06]
  - C04 -4 character/digits
  - C05 -5 character/digits
  - C06 -6 character/digits
- NUMBER:BASE=B10 -String base [B10]
  - B10 -Base 10 (decimal)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.5.19 OBJECT *BCD*

SYNOPSIS

```
HELP:   Use BCD format for the length field
PATH:   XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT
TYPE:   OBJECT
SYNTAX: BCD(WIDTH=U16/U24/U32,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in BCD format (4 Bit are one decimal digit). You can define the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:WIDTH=U16/U24/U32 -Bit width/rate [B32]
  - U16 -16 bit unsigned BCD (POV with 4 digits)
  - U24 -24 bit unsigned BCD (POV with 6 digits)
  - U32 -32 bit unsigned BCD (POV with 8 digits)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.5.20 OBJECT *HOST*

SYNOPSIS

```
HELP:   Use host format (LLxx) for the length field
PATH:   XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT
TYPE:   OBJECT
SYNTAX: HOST(ENDIAN=SYSTEM/BIG/LITTLE,EXCLUDE)
```

DESCRIPTION The object defines the 4 byte host specific format (LLxx) of the length field. This format uses the first 16 bits as integer in big endian or little endian byte order to represent the record length including or excluding the record length field itself and the last 16 bits are set to zero when writing and ignored when reading.

For example, a record with 136 bytes content has a default length field (big endian, inclusive) of:

```
x'008D0000'
```

The default conforms to the length fields used by the data management system of MVS.

ARGUMENTS

- NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [BIG ENDIAN] for first 16 Bit
  - SYSTEM -Byte order of the current system
  - BIG -Big endian byte order
  - LITTLE -Little endian byte order
- SWITCH:EXCLUDE -Record length field not part of the length value [OFF]

4.5.21 OBJECT *SUBSYS*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   XCHK.INPUT.SAV.FILE.FIO.REC
TYPE:   OBJECT
SYNTAX: SUBSYS(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
SUBSYS(NAME='name' parm='keyword,keyword=value,value')
SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- STRING:NAME='str' -Name of the subsystem
- STRING:PARM='str' -Parameter list for the subsystem (comma separated)

4.5.22 OBJECT *TXT*

SYNOPSIS

```
HELP:   Read a file text-oriented (mainly for open world)
PATH:   XCHK.INPUT.SAV.FILE.FIO
TYPE:   OBJECT
SYNTAX: TXT(NAME='str'/STREAM/DUMMY,RECMOD=STOP/CUT/WRAP,RECLEN=num,RECCNT=num,PRNOUT=num, ↩
    SUBSYS(),REMOVE)
```

DESCRIPTION This object defines the text-oriented read operation for files. This method reads text records from a file with text delimiters. The data must be a text file in the platform specific character set where each record is terminated with the system specific delimiter. You can define the maximum record length and what to do, if a record is longer than the provided record length.

On read, you can also activate the conversion of the data into a printout-like amount of records, i.e. print control characters are replaced by records so that it looks like the data was printed.

ARGUMENTS

- STRING:NAME='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- NUMBER:RECMOD=STOP/CUT/WRAP -Mode used to read records [STOP]
  - STOP -Stop if record too long
  - CUT -Cut if record too long
  - WRAP -Wrap if record too long
- NUMBER:RECLEN=num -Record (Host) length /Line (Unix/Win) length for read operations [512]
- NUMBER:RECCNT=num -Amount of records handled in one read operation [128]
- NUMBER:PRNOUT=num -Amount of lines per page if print control replacement active [0 = inactive]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.5.23 OBJECT *SUBSYS*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   XCHK.INPUT.SAV.FILE.FIO.TXT
TYPE:   OBJECT
SYNTAX: SUBSYS(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- STRING:NAME='str' -Name of the subsystem
- STRING:PARM='str' -Parameter list for the subsystem (comma separated)

4.5.24 OBJECT *FL4*

SYNOPSIS

```
HELP:   Read a FLAMFILE record-oriented
PATH:   XCHK.INPUT.SAV.FILE.FIO
TYPE:   OBJECT
SYNTAX: FL4(NAME/FLAMFILE='str',MEMBER='str',RECLEN=num,RECCNT=num,TRUCAT,NONFIL,PRNCTR= ↩
    DETACH/RETAIN/ERASE,DEVICE=FILE/USER,PASSWD='bin'/DEFAULT,KMELIB='str',KMEFUC='str', ↩
    KMEPAR='str',EXTCMP='str',EXTDCO='str',SECINF=YES/NO/IGNORE/MEMBER,REMOVE)
```

DESCRIPTION This object defines the FLAM4 read operation for files. This method reads records from a FLAMFILE of version 4 or older. It uses the FLAM4 record interface to read members from a FLAMFILE. The parameters below are explained in detail by the FLAM command of FLCL.

ARGUMENTS

- STRING:NAME/FLAMFILE='str' -Name/URL of FLAMFILE to read [required]
- STRING:MEMBER='str' -Name or index of the member in the FLAMFILE
- NUMBER:RECLEN=num -Maximal record length for read operations [512]
- NUMBER:RECCNT=num -Amount of records handled in one read operation [128]
- SWITCH:TRUCAT -Allow truncation of records [FALSE]
- SWITCH:NONFIL -Switch to replace original file name with FILEnnnn [FALSE]
- NUMBER:DEVICE=FILE/USER -Device used to store the FLAMFILE [FLATFILE]
  - FILE -Write to a file
  - USER -Use user IO
- STRING:PASSWD='bin'/DEFAULT -Passphrase to decrypt the FLAMFILE [”]
  - DEFAULT -FLAM4 default password
- STRING:KMELIB='str' -Library name for FKME [”]
- STRING:KMEFUC='str' -Function name of FKME [”]
- STRING:KMEPAR='str' -Parameter for the call of FKME [”]
- STRING:EXTCMP='str' -Name of user exit for compression
- STRING:EXTDCO='str' -Name of user exit for decompression
- NUMBER:SECINF=YES/NO/IGNORE/MEMBER -Define handling of secure information
  - YES -Create secure information at compression
  - NO -Do not store any additional secure information
  - IGNORE -Ignore any security violations on decompression
  - MEMBER -Only member-specific security informations are verified
- SWITCH:REMOVE -Remove FLAMFILE after successful processing [FALSE]

4.5.25 PARAMETER *PRNCTR*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   XCHK.INPUT.SAV.FILE.FIO.FL4
TYPE:   NUMBER
SYNTAX: PRNCTR=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.5.26 OVERLAY *CNV*

SYNOPSIS

```
HELP:   Conversion [NONE]
PATH:   XCHK.INPUT.SAV.FILE
TYPE:   OVERLAY
SYNTAX: CNV[{BLK()/REC()/GZP()/BZ2()/LXZ()/ZIP()/CHR()/BAS()/HSH()}...]
```

DESCRIPTION With the overlay CNV you can choose the different kind of conversions supported at read operations.

4.5.27 OBJECT *BLK*

SYNOPSIS

```
HELP:   Convert a record list to one block (binary or text) of data
PATH:   XCHK.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: BLK(BUFSIZ=num,METHOD=HOST/BIN/REC/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM ↩
    /ORIGINAL,CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE ↩
    /UTF32BE/UCS4LE/UTF32LE,SUPTWS,RPLTAB/RPLHTB=num,RPLVTB=num,RPLCTR=SPACE/SUBSTITUTE/ ↩
    DELETE)
```

DESCRIPTION The block converter can be used to convert records into data blocks with delimiters. On output, it can also add different length formats in front of the records. This conversion component uses the text, block and binary formatting components and provides all their features.

ARGUMENTS

- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:METHOD=HOST/BIN/REC/NL/USS/LF/UNIX/CR/OLDMAC/CRLF/WINDOWS/DLM/SYSTEM/ORIGINAL -Method for block/text formatting [SYSTEM]
  - HOST -Adds no delimiter (HOST)
  - BIN -Adds no delimiter (HOST)
  - REC -Adds no delimiter (HOST)
  - NL -Adds delimiter new line (USS)
  - USS -Adds delimiter for USS (new line)
  - LF -Adds delimiter line feed (UNIX)
  - UNIX -Adds delimiter for UNIX (line feed)
  - CR -Adds delimiter carriage return (MAC)
  - OLDMAC -Adds delimiter for old MACs (carriage return)
  - CRLF -Adds delimiter carriage return line feed (WIN)
  - WINDOWS -Adds delimiter for WINDOWS (carriage return line feed)
  - DLM -Adds the system specific delimiter (DEFAULT)
  - SYSTEM -Adds the system specific delimiter (DEFAULT)
  - ORIGINAL -Adds the original data at the end of a line (only for band FMT)
- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- SWITCH:SUPTWS -Suppress trailing whitespace [FALSE]
- NUMBER:RPLTAB/RPLHTB=num -Replace horizontal tabulators by spaces using this tab width [0 =no replacement]
- NUMBER:RPLVTB=num -Replace vertical tabulators by new lines using this tab width [0 = no replacement]
- NUMBER:RPLCTR=SPACE/SUBSTITUTE/DELETE -Replace remaining control characters [NONE]
  - SPACE -Replace control characters with whitespace character (0x20/0x40)
  - SUBSTITUTE -Replace control characters with substitution character (0x1A/0x3F)
  - DELETE -Remove control characters

4.5.28 OBJECT *REC*

SYNOPSIS

```
HELP:   Convert a block (binary or text) of data into a list of records
PATH:   XCHK.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: REC(METHOD=BIN/DLM/REC/L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX,CHRSET=NONE/ ↩
    SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/ ↩
    UTF32LE,RPLFFD=num,RPLTAB/RPLHTB=num,RPLVTB=num,RPLCTR=SPACE/SUBSTITUTE/DELETE,SUPTWS, ↩
    NELDLM,PADCHR=num,RECLEN=num,BUFSIZ=num,INICNT=num,SKPTXT)
```

DESCRIPTION The record converter can be used to convert data blocks with delimiters or on input with different record length formats into data lists with records. This conversion component uses the text and record formatting component and provides all its features.

Auto detection of 4 byte record length fields is attempted if no method is provided. If the data block contains no record length fields and the switch to skip text formatting is not enabled, then all capabilities of the text formatting component are used to form records.

The SKPTXT flag causes input blocks to be only converted to records if record length fields are detected. This can be useful, for example, for reading ZIP archives containing record-oriented files. PKZIP stores record-oriented files with a 4 byte length field (length of the field itself not included) in front of the record data in little endian (L4X). Another example files stored with FILE-DATA=RECORD (B4X) in USS. When reading such a file, the length fields must be interpreted to re-build the corresponding records. This conversion can be added after any I/O or other conversion step.

ARGUMENTS

- NUMBER:METHOD=BIN/DLM/REC/L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX -Method fo r record/text formatting [DEFAULT]
  - BIN -Wrap data block in records of record length
  - DLM -Parse data block for text delimiter (DEFAULT)
  - REC -Use read record length fields (only useful on mainframes)
  - L4I -Parse data based on 4 byte length fields:Little endian integer, length inclusive
  - L4X -Parse data based on 4 byte length fields:Little endian integer, length exclusi ve (ZIP)
  - B4I -Parse data based on 4 byte length fields:Big endian integer, length inclusive
  - B4X -Parse data based on 4 byte length fields:Big endian integer, length exclusive (USS)
  - S4I -Parse data based on 4 byte length fields:System endian integer, length inclusive
  - S4X -Parse data based on 4 byte length fields:System endian integer, length exclusi ve (VAR)
  - HLI -Parse data based on 4 byte length fields:Little endian short (LLxx), length in clusive
  - HLX -Parse data based on 4 byte length fields:Little endian short (LLxx), length ex clusive
  - HBI -Parse data based on 4 byte length fields:Big endian short (LLxx), length inclu sive (MVS)
  - HBX -Parse data based on 4 byte length fields:Big endian short (LLxx), length exclu sive
  - HSI -Parse data based on 4 byte length fields:System endian short (LLxx), length in clusive
  - HSX -Parse data based on 4 byte length fields:System endian short (LLxx), length ex clusive

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assuming n lines per page [0 =no replacement]
- NUMBER:RPLTAB/RPLHTB=num -Replace horizontal tabulators by spaces using this tab width [0 =no replacement]
- NUMBER:RPLVTB=num -Replace vertical tabulators by new lines using this tab width [0 = no replacement]
- NUMBER:RPLCTR=SPACE/SUBSTITUTE/DELETE -Replace remaining control characters [NONE]
  - SPACE -Replace control characters with whitespace character (0x20/0x40)
  - SUBSTITUTE -Replace control characters with substitution character (0x1A/0x3F)
  - DELETE -Remove control characters
- SWITCH:SUPTWS -Suppress trailing whitespaces [FALSE]
- SWITCH:NELDLM -Activate NEL (0x85) as delimiter for ASCII character sets [FALSE]
- NUMBER:PADCHR=num -Padding character [0x00]
- NUMBER:RECLEN=num -Length used to cut the data block in records [512]
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of records for preallocation [128]
- SWITCH:SKPTXT -Convert only if record length fields found in the data [FALSE]

4.5.29 OBJECT *GZP*

SYNOPSIS

```
HELP:   Read ZLIB/GZIP/RCF1950-2 compressed data
PATH:   XCHK.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: GZP(MEMBER='str',BUFSIZ=num)
```

DESCRIPTION The GZIP converter can be used to compress or decompress data streams.

The result of a compression is a binary block. The result of a decompression can be text, XML, a base encoding or binary data. Record boundaries are lost. Please use delimiters if records must be exchanged.

The GZIP decompression can read all supported formats of zlib and writes RFC1952/51/50-compliant data streams. If a IBM ZEDC acceleration card is available and can be used, the ZEDC is used to save CPU time. In this case, however, the compression does not conform to RFC1951. The compressed data is encoded based on RFC1950 and the file format conforms to RCF1952.

For a checksummed GZIP header, an additional switch must be activated, because this feature is not supported by all GZIP implementations. The GZIP header is used to store meta information about the data, which can be used if FLAM is also used for decompression. This can be very useful to clearly define the character set, CCSID and other information about the data.

If no compression level is set and the data is marked as being without redundancy, then level 0 (copy) is used. Otherwise, the default level 6 is used.

Decompression of concatenated GZIP files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
filename/#memberindex

test.gz/#3
test.gz/:2
```

GZIP data streams are written to ZIP archives as compression mode 8. The GZIP header and trailer are replaced by a ZIP local file header. If a deflated file (compression mode 8) is read from a ZIP archive, it is converted to a GZIP data stream which can be inflated with this component.

ARGUMENTS

- `STRING:MEMBER='str' -Index of the member in the GZIP-File to read`
- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation[65536]`

4.5.30 OBJECT *BZ2*

SYNOPSIS

```
HELP:   Read BZIP2 compressed data
PATH:   XCHK.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: BZ2(MEMBER='str',BUFSIZ=num,SMLSPC)
```

DESCRIPTION The BZIP2 converter can be used to compress or decompress data streams compatible with the BZIP2 utility on UNIX systems.

The result of a compression is a binary block. The result of a decompression can be text, XML, a base encoding or binary data. Record boundaries are lost. Please use delimiters if records must be exchanged.

There is no header to store meta information about the data, so all metadata is lost.

If no compression level is defined, the level used depends on the block size. A block size larger than 900KB results in the best compression level 9, greater than 800KB and smaller than 900KB in level 8, and so on. For blocks smaller than 100KB level 1 is used.

Decompression of concatenated BZIP2 files is also possible. All concatenated files will be decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
filename/:memberindex
filename/#memberindex

test.bz/#3
test.bz/:2
```

BZIP2 data is written to ZIP archives as compression mode 12. The BZIP2 header and trailer are replaced by a ZIP local file header. If a BZIP2 file (compression mode 12) is read from a ZIP archive, it is converted to a BZIP2 data stream and can be decompressed with this component.

ARGUMENTS

- `STRING:MEMBER='str' -Index of the member in the BZIP2-File to read`
- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]`
- `SWITCH:SMLSPC -Use small memory space for decompression [FALSE]`

4.5.31 OBJECT *LXZ*

SYNOPSIS

```
HELP:   Read XZ (LZMA) compressed data
PATH:   XCHK.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: LXZ(MEMBER='str',BUFSIZ=num)
```

DESCRIPTION The LZMA/XZ converter can be used to compress or decompress data streams compatible with the XZ utility on UNIX systems.

The result of a compression is a binary block. The result of a decompression can be text, XML, a base encoding or binary data. Record boundaries are lost. Please use delimiters if records must be exchanged.

There is no header to store meta information about the data, so all metadata is lost.

If no compression level is defined and the data is marked as being without redundancy, then level 0 (copy) is used. In all other cases the default level 6 is used.

Decompression of concatenated XZ files is also possible. All concatenated files are decompressed into one file. It is also possible to decompress one specific member of this concatenation by specifying the member index of the file to be decompressed:

```
   filename/:memberindex
   filename/#memberindex

   test.xz/#3
   test.xz/:2
```

XZ data streams are written to ZIP archives as compression mode 14 (LZMA). The XZ header and trailer are replaced by a ZIP local file header. If an LZMA file (compression mode 14) is read from a ZIP archive, it is converted to an XZ data stream which can be decompressed with this component.

ARGUMENTS

- `STRING:MEMBER='str' -Index of the member in the XT-File to read`
- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]`

4.5.32 OBJECT *ZIP*

SYNOPSIS

```
HELP:   Read compressed data (GZIP, BZIP2, LZMA, ...)
PATH:   XCHK.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: ZIP(MEMBER='str',BUFSIZ=num)
```

DESCRIPTION The ZIP converter can read compressed data streams. The compression algorithm is detected automatically. All compression algorithms (GZIP, BZIP2, XZ,...) are supported which are also available separately on the conversion level of XCNV. This component simply uses the dedicated compression components and adds the auto detection. Please refer to each individual compression component for more details.

This component can be used together with FIO.ZIP() to decompress members which were read from a ZIP archive as FIO.ZIP() does not provide any compression functionality by itself.

```
   inp(sav.fil(fio.zip(name=my.zip member=test.txt) cnv.zip()))
```

ARGUMENTS

- `STRING:MEMBER='str' -Index of the member in the compressed file to read`
- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation[65536]`

4.5.33 OBJECT *CHR*

SYNOPSIS

```
HELP:   Convert character sets (ASCII, EBCDIC, UNICODE)
PATH:   XCHK.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: CHR(BUFSIZ=num,RECCNT=num,METHOD=AUTO/BLOCK/RECORD/KEEP/SKIP,CASMOD=UPPER/LOWER/ ↩
    FOLD/SUPPER/SLOWER/USRTAB,FROM='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,MODE=STOP/IGNORE ↩
    /SUBSTITUTE/IDENTITY,TO='str'/DEFAULT/ASCII/EBCDIC,WRTBOM,HDLBOM/BOM,ENL2LF,ELF2NL, ↩
    SUBCHR[num/SYSTEM...],SYSTAB=ICONV,USRTAB='str'/NPAS/SEPA/DELA/DLAX,REPFIL='str'/STDOUT/ ↩
    STDERR,SKPBIN,SKPEQU)
```

DESCRIPTION This component can be used to convert a text stream to a different character set after reading or before writing. Character conversion requires a source CCSID (FROM) and a target CCSID (TO). Using the keyword MODE, you can define the behavior when invalid or incomplete characters are encountered in the input data. It is possible to stop at, ignore or substitute unsupported characters. For the latter case, you can define a substitution character list (SUBCHR) and/or a user substitution table (USRTAB) and/or use one of the pre-defined system substitution tables (SYSTAB) for transliteration. When using a system table, you can override some of the pre-defined code points using a user table. Each character must be defined as UNICODE point in hexadecimal notation.

Each ignore or substitution process, including transliteration, can be recorded in a report file for review. Moreover, you can convert to upper, lower, special case or use case folding. The component also provides powerful support for byte order marks (BOM) including:

- Byte order change handling at read (for example UTF-16LE in UTF-16BE)
- Byte order marks per stream (if block oriented) or record at write
- Charset detection based on byte order marks

You can differentiate between block- and record-oriented conversion by the keyword METHOD.

- record oriented conversion - throws an error if an incomplete byte sequence occurs at the end of a record
- block oriented conversion - no check for record boundaries, regards all data as one block

If the input text is record-oriented then record orientation will be used by default, and you must explicitly enable block-oriented conversion if you do not intend to check the data at each end of a record. If you choose block orientation, the record lengths table will be lost. Additionally, you can enforce to skip incomplete character sequences at the end of a record to keep the length information.

In case of an error, all position and counter values printed to the error trace contain the current indexes. All indexes start at zero. The first block or record is unit 0.

It supports replacement of EBCDIC new line (0x15) characters by line feed (0x25) at character conversion takes place. When converting to UTF-8/16/32, this will result in the new line characters to be converted to line feeds (0x0A) instead of UTF-8 next line characters (0xC285).

In read mode, the component supports auto-detection of charsets. This can be activated by using the keyword DEFAULT as CCSID. Auto-detection will analyze the first block of input data to determine if the input uses ASCII, EBCDIC or a UTF encoding. The statistic-based computation cannot determine the actual ASCII or EBCDIC character set. Instead, the final CCSID is derived from the language identifier of the environment variable LANG.

Character conversion can skipped or enforced if the CCSIDs are equal or or binary data is detected.

To get a list of supported CCSIDs, use the command below:

```
   flcl INFO GET.CCSID
```

For more information, please have a look at the man pages for each parameter.

ARGUMENTS

- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]`
- `NUMBER:RECCNT=num -Initial amount of records for preallocation [128]`

4.5.34 PARAMETER *METHOD*

SYNOPSIS

```
HELP:   Enforces record or block-oriented conversion [AUTO]
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: METHOD=AUTO/BLOCK/RECORD/KEEP/SKIP
```

DESCRIPTION The method of character conversion defines how the input data fields are handled. A data field can be one block or a number of records dependent from how the data is read and converted before character conversion is done. Is the data field a list of records then the record oriented method shall be used. Is the data field a single block the block oriented conversion shall be used. This means that the METHOD must only be set to enforce record or block orientation against the usual way of processing the data.

- Block Orientation - This means all records are handled as one block and the record length information will go lost. If incomplete characters occur at the end of this block it will be remembered as rest and copied to the next block. Using this method there will be no interruption of the data stream.
- Record Orientation - This method ensures that each record contains only complete character sequences. An incomplete character sequence will only be accepted at the end of the last record and handled also as rest for the record in the next data field. To enforce record orientation processing of one block of data there will be no difference to block orientation conversion, besides for processing the last block.

Thus it makes only sense to switch for a record organized data field to block orientation conversion. But this is very critical, as the length information will go lost. This means that there will be no chance to retrieve the record information, besides the record contains some delimiters which can be used to build a new list of record lengths. So we recommend to use the METHOD switch only if you know what you are doing. In some cases analysis of the error trace might result in the correct method that can be used to convert the text.

If an incomplete byte sequence progresses in the next record, then it can be helpful to enforce block oriented conversion, because the record length conversion can be invalid. On the other side, if the next record starts with a new valid byte sequence, then a skip of the damaged byte sequence in the last record can be helpful.

If SKIP is selected then each incomplete character at the end of a record is recorded in the report file with a special code point 0xFFFFFF including the really skipped byte sequence. If this problem happens in the last block you can enforce record oriented

conversion with SKIP to convert the data and the incomplete rest of the last block. They are written to the report file. Normally the corresponding MODE is used for the final block to prevent a rest.

Please don't mix up SKIP as METHOD for the rest of a record contained in a data field and the IGNORE as MODE for the content of a record as unit of work.

The default method depends on the data. If the data is marked as block-oriented or the record type is text with delimiters, then the method BLOCK is used. In all other cases, record orientation with the method KEEP is used.

SELECTIONS

- `AUTO -Automatic orientation (block or record with keep)`
- `BLOCK -Block-oriented conversion (all data is one chunk)`
- `RECORD -Record-oriented conversion (check record boundaries)`
- `KEEP -Keep (Move a rest from the end to begin of next record)`
- `SKIP -Skip a rest at the end of a record (not recommended)`

4.5.35 PARAMETER *CASMOD*

SYNOPSIS

```
HELP:   Define upper, lower or special case conversion [NONE]
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: CASMOD=UPPER/LOWER/FOLD/SUPPER/SLOWER/USRTAB
```

DESCRIPTION With the case modifier, you can perform conversion to upper or lower case as well as apply case folding and some special case conversions.

Case folding converts all characters to lower case but is more compatible with language specific rules for doing so. For example, this may be useful for case-insensitive string comparison if the strings contain non-Latin characters (code point < 128).

This mapping is done for each code point in front of substitution or transliteration. The respective case is accomplished on the basis of a separate Unicode conversion table in memory. This conversion can be manipulated if user mapping is activated and mapping definitions (⋆) are defined through the provided user table.

The options UPPER and LOWER convert characters without expansion, i.e. returned strings will never grow in length. Note, however, that case folding and the special options (convert e.g. ß to SS) can result in an expansion of data.

All these conversions are pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `UPPER -Change character to upper case`
- `LOWER -Change character to lower case`
- `FOLD -Change character to folded case (for compare)`
- `SUPPER -Change character to special upper case`
- `SLOWER -Change character to special lower case`
- `USRTAB -Change character according to user table`

4.5.36 PARAMETER *FROM*

SYNOPSIS

```
HELP:   Conversion from this CCSID [auto detection]
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   STRING
SYNTAX: FROM='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- DEFAULT –Use default CCSID (auto-detect)
- ASCII –Use default ASCII CCSID (environment)
- EBCDIC –Use default EBCDIC CCSID (environment)
- BOMUTF –Determine the correct UTF CCSID from byte order mark (BOM)
- BOMUCS –Determine the correct UCS CCSID from byte order mark (BOM)

4.5.37 PARAMETER *MODE*

SYNOPSIS

```
HELP:   Define handling of non-convertible characters
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: MODE=STOP/IGNORE/SUBSTITUTE/IDENTITY
```

DESCRIPTION With the MODE selection you can define how an invalid or incomplete character is handled inside of a unit by this component. All possible modes are listed at the end of this chapter.

This MODE is valid for the content of each unit (block or record). At the end of a unit there can be an incomplete character. Such a rest will be handled dependent on the METHOD you choose. The rest of the last unit of a data field will be concatenated to the first unit of the next data field. For all other records in a data field the handling of such a rest will be defined by the respective METHOD.

The SKIP case for METHOD is only relevant for incomplete byte sequences at the end of a unit. The ignore and substitution mode is only effective for all characters in the middle of the unit, besides an incomplete byte sequence occurs in the last unit. Please don't mix up the IGNORE case for MODE with the SKIP case of METHOD.

If a system or user substitution table is used the stop, ignore or substitute are only relevant for characters not managed by the CCSIDs and the substitution tables. This means that the USRTAB and SYSTAB are added to the capabilities of character conversion and only for the rest the chosen MODE is used.

4.5.37.1 CONSTANT *STOP*

SYNOPSIS

```
HELP:   Stop at the first non-convertible character
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR.MODE
TYPE:   NUMBER
SYNTAX: STOP
```

DESCRIPTION Stops conversion at the first incomplete or invalid character which can not be converted. An error message is given with the position of abortion and the value in the input stream. An entry will be (STP()) written to the report file to log this event. To be compatible with other conversion tools, this is the default case, but we recommend to use IGNORE or SUBSTITUTE to prevent any abortion.

indexterm:[Constant STOP}

4.5.37.2 CONSTANT *IGNORE*

SYNOPSIS

```
HELP:   Ignore non-convertible characters
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR.MODE
TYPE:   NUMBER
SYNTAX: IGNORE
```

DESCRIPTION Ignores incomplete or invalid characters and writes a report entry for each character. IGNORE is a substitution by no corresponding character that means no substitution table is required.

indexterm:[Constant IGNORE}

4.5.37.3 CONSTANT *SUBSTITUTE*

SYNOPSIS

```
HELP:   Substitute non-convertible characters
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR.MODE
TYPE:   NUMBER
SYNTAX: SUBSTITUTE
```

DESCRIPTION Substitutes incomplete or invalid character and writes a report entry for each character. For SUBSTITUTE you can define a substitution character list (SUBCHR). If no SUBCHR is provided then this option substitutes invalid or incomplete characters with code point 0x00001A. If a SUBCHR is defined all invalid or incomplete characters will be substituted by the defined list of code points that can be up to 8. If you specify a SUBCHR the substitution mode will be enabled.

indexterm:[Constant SUBSTITUTE}

4.5.37.4 CONSTANT *IDENTITY*

SYNOPSIS

```
HELP:   Copy non-convertible characters (supported only for single byte conversions)
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR.MODE
TYPE:   NUMBER
SYNTAX: IDENTITY
```

DESCRIPTION Copies invalid characters from input to output and writes a report entry for each character. Resulting output could be not printable.

indexterm:[Constant IDENTITY}

4.5.38 PARAMETER *TO*

SYNOPSIS

```
HELP:   Conversion to this CCSID [UTF-8]
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   STRING
SYNTAX: TO='str'/DEFAULT/ASCII/EBCDIC
```

DESCRIPTION The TO code defines the encoding string or CCSID for the output text stream. The default value of this CCSID for read operations is UTF-8. For write operations, the default value is derived from the environment variable LANG. If LANG is not set, the default ASCII or EBCDIC system-codepage is used, depending on the platform.

To get a list of supported encoding strings, please use the command below:

```
   flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs please use:

```
   flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
   CCSID='37'    will be equal to CCSID='IBM037'
   CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit. Optionally, you can use one of the keywords below to use the system-specific default ASCII or EBCDIC CCSID.

SELECTIONS

- DEFAULT –Use default CCSID (UTF-8)
- ASCII –Use default ASCII CCSID (environment)
- EBCDIC –Use default EBCDIC CCSID (environment)

4.5.39 PARAMETER *WRTBOM*

SYNOPSIS

```
HELP:   Write byte order mark in front of the output text
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: WRTBOM
```

DESCRIPTION If the write byte order mark keyword is defined, then the corresponding BOM sign is written at the start of the output stream (block oriented) or at the beginning of each record. If this BOM keyword is not used, then the BOM sign will not be written (this is valid independent of the chosen encoding/CCSID). If a valid BOM sign is located in the middle of the input stream, it is removed from the data. If you define the BOM keyword and the target charset contains only single bytes or no valid BOM sign, then no byte order mark will be written to the output file.

If an invalid BOM sign is recognized, then it will be ignored and causes no entry in the report file. If the byte order changes in the middle of text and the BOM handling switch is not activated, the conversion will stop and an error message is thrown.

By default, the write BOM keyword is set to OFF.

4.5.40 PARAMETER *HDLBOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: HDLBOM/BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.5.41 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.5.42 PARAMETER *ELF2NL*

SYNOPSIS

```
HELP:   Replace EBCDIC line feed (0x25) by new line (0x15) on output
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: ELF2NL
```

DESCRIPTION The replacement of EBCDIC line feed control characters (0x25) by new line characters (0x15) at conversion to EBCDIC can be activated on non EBCDIC (ASCII) systems or deactivated on EBCDIC systems by enabling this switch. This is for example required to prepare text files on open platforms for USS on z/OS. The usual delimiter for text in ASCII and UTF is 0x0A and will normally be translated to 0x25 in EBCDIC. On z/OS(USS), however, 0x15 (NL) is expected.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

To ensure a correctly delimited text record, only a line feed and not a carriage return (0x0D) line feed (0x0A) is used. If text formatting is used with method=NL, this will result in a single LF on distributed systems and the internal conversion will be activated by default to produce a valid text file for z/OS under USS on Windows, Unix and other open platforms.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.5.43 PARAMETER *SUBCHR*

SYNOPSIS

```
HELP:   Default substitution character list [0x1A] (maximal 8 code points)
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: SUBCHR[num/SYSTEM...]
```

DESCRIPTION As substitution character (SUBCHR) you can define a list with up to 8 UNICODE points which are used for the substitution of invalid or incomplete characters if the corresponding code point is not defined in the substitution table (see SUBTAB). If you assign to it the keyword SYSTEM the value 0x1A is written as default substitution character.

If you activate substitution and no SUBCHR is defined, then unsupported characters will be replaced by CODEPOINT 0x00001A.

On the other hand SUBCHR will replace every undefined code point in the substitution table, what is the same behavior when using a USRTAB with corresponding SUBCHR values for every possible code point (000000-200000). If SUBCHR but no MODE is defined the substitution mode will be activated.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `SYSTEM -Use system specific substitution char (0x1A for ASCII/UTF, 0x3F for EBCDIC)`

4.5.44 PARAMETER *SYSTAB*

SYNOPSIS

```
HELP:   Define a preloaded system substitution table
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   NUMBER
SYNTAX: SYSTAB=ICONV
```

DESCRIPTION With the system table (SYSTAB) you can load a substitution table with a predefined transliteration schema. Entries in the USRTAB can be used to override predefined transliteration entries or add missing transliteration code points. The options listed at the end of this chapter are available.

If a substitution table is used then the MODE option (STOP/IGNORE/SUBSTITUTE) is only relevant for characters that cannot be converted or substituted.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

SELECTIONS

- `ICONV -Preload with ICONV//TRANSLIT substitution table`

4.5.45 PARAMETER *USRTAB*

SYNOPSIS

```
HELP:   Name of the file containing the user conversion table
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   STRING
SYNTAX: USRTAB='str'/NPAS/SEPA/DELA/DLAX
```

DESCRIPTION With this parameter, you can choose a predefined subset or standard manipulation for the system tables via keyword (see `flcl help "xcnv.inp .sav.fil.cnv.chr.usrtab"` for available keywords). Alternatively, you can use a file containing a user defined substitution table (USRTAB).

For the user table file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   USRTAB=NPAS                          ; use XOEV predefined subset
   USRTAB=DELA                          ; German Latin single byte subset
   USRTAB='~.MYUSRTAB(USRTAB1)'         ; PDS on z/OS
   USRTAB='<SYSUID>.MYUSRT02'           ; PS dataset on z/OS
   USRTAB='~/myusrtab3.txt'             ; Unix path name
   USRTAB='<HOME>\usrtab\myusrtab4.txt' ; Windows path name
```

A user defined substitution table has the syntax listed below:

```
   usr_tab_file         -> usr_definition_list
   usr_definition_list  -> usr_definition usr_definition_list
                        |  EMPTY
   usr_definition       -> '(' codepoint_definition ')'
   codepoint_definition -> cp_activation
                        |  cp_deactivation
                        |  cp_transliteration
                        |  cp_standard_mapping
                        |  cp_usr_case_mapping
   cp_activation        -> ['+'] CODEPOINT  [ '-' CODEPOINT ]
   cp_deactivation      ->  '-'  CODEPOINT  [ '-' CODEPOINT ]
   cp_transliteration   -> ['+'] CODEPOINT '=' code_point_list
   cp_standard_mapping  ->  '*'  CODEPOINT '=' code_point_list
   cp_usr_case_mapping  ->  '^'  CODEPOINT '=' code_point_list
   code_point_list      -> CODEPOINT separator code_point_list
                        |  CODEPOINT
                        |  EMPTY
   separator            -> '/'
```

```
                        |  ',' 
                        |  ';'
```

Note: In the description below "/" is used as code point separator.

The optional signs in front of a code point (CP) have the following meaning:

```
  '+' - a valid transliteration (only if CP not in the target set)
  '*' - a valid mapping         (this translation is always done)
  '^' - a case mapping/folding  (only if case=usrtab is activated)
  '-' - an invalid definition   (removes CP for subset definitions)
```

If no sign is used, then a transliteration for a valid code point is defined. Valid code points are accepted, invalid code points result in in an error. Error handling depends on the mode (STOP,IGNORE, SUBSTITUTE). Definition of + code points without code point list, thus without = is equal definition of valid code points. With an asterisk *, you can define an enforced mapping for this code point. For example, this can be used to delete this character if no code point list is provided or to translate this character always in another value. This can also be used to convert code points outside of a subset into this subset.

In contrast to the transliteration (+ is only done if a character doesn't exist in the target encoding), the mapping is always done for each character. Invalid (−) code points can be used to deactivate a code point. To activate (+) or deactivate (−) a range of code points you can use the minus mark (−) followed by one sub code point. The range goes from the smaller code point to the bigger one. For example:

```
  (-00-7F) deactivate all US-ASCII code points
  (+39-30) activate all decimal digits
```

The range operator (−) with the optional plus sign (+) can also be used to activate code points.

To define an own case mapping or folding (required for certain subsets) the caret sign (^) can be used. This mapping is only done if the user table is activated for case mapping (CASE=USRTAB).

To define transliterations/substitutions/mappings an assignment (=) of a code point list is required. The code point list can contain a maximum of 8 code points. If no code point list is specified, then the character is ignored. A transliteration to itself simply activates this code point. The same is valid for mapping definitions. A code point is a hexadecimal number representing a 21 bit UNICODE point.

```
  00000000 to 001FFFFF hex
```

If a code point is not defined in the substitution table (USRTAB and/or SYSTAB), the appropriate substitution character (SUBCHR) is used. If SUBCHR is not set, no substitution is performed. Depending on the MODE, character conversion stops with an error (STOP) or ignores the character (IGNORE).

Text before and after brackets are comments.

Between "(" and "=" or "/" or ")" you can define hex digits until the first non-hex digit. All non-hex digits up to the next separator are interpreted as comment. Leading whitespace is ignored.

```
  REPLACE GERMAN SZ (00DF=0073/0073)                  WITH ss
THIS IS AN EXAMPLE FOR COMMENTS IN CODE POINT LIST:
  REPLACE EURO MARK (20AC= 45#E / 55#U / 52#R / 4F#O) WITH EURO
  REPLACE BOM MARK  (EFFF=)                           WITH NOTHING
```

Leading zeros are allowed. If you don't supply a hex value, then 0x00 is used.

```
  REPLACE GERMAN SZ (00DF=/)                   WITH 0x00 0x00
  REPLACE EURO MARK (20AC=00000045/00055/000052/000004F) WITH EURO
```

ATTENTION: Please don't use parentheses "()" or the operators in your comments.

To describe your own subsets, you can use a user table without a system table. The USRTAB can also be used to overwrite or add transliterations when a system substitution table (for example SYSTAB=ICONV) is used. The transliteration works recursively, that is if one of the substitution code points is not in the target set, this substitution will be used instead and so on.

```
REPLACE GERMAN OE (0000D6=004F/0045)       WITH O E
REPLACE GERMAN SZ (0000DF=0073/0073)       WITH s s
REPLACE EURO MARK (0020AC=00D6/00DF/52/4F) WITH OE SZ R O
```

If you have a EURO sign in your text and convert it to *Latin1*, the resulting byte string will be *D6DF524F*. On the other hand, if you convert it to *ASCII*, the byte string will be *4F457373524F*.

The mapping as described above is always done and is also done recursive including the transliteration result. For example, if you define a mapping `(*30=39)` then the target data won't have any zero in the resulting text.

By replacing other code points recursively, you could easily cause infinite loops. To prevent this, the amount of replacements and the length of one replacement is limited to a maximum of 64. Note that this could still result in a large expansion of data. Therefore, be careful about defining recursive substitutions.

A sample user table for the ICONV system table to change the transliteration of German umlauts to AE, OE or UE is located in the SAMPLE directory under CCUTDEXL. Another sample user table called CCUTNPAS defines the *string.latin* subset (XOEF) which is mainly used for statutory reporting.

The order of definition is the order of processing the definitions. For exampel, if you define a mapping or a transliteration for code point X and later you make this code point invalid, then the code point is invalid. On the other hand, you can first deactivate all code points, then activate your subset and define your transliterations (best fit) or mappings.

For example: To define a non-expansion best fit mapping a user table can be used to delete all combined characters for single byte code pages. For the ICONV system transliteration table, a sample user table can be found under the name CCUTBFM1.

Some other subset user table definitions were added. For example:

- currently valid SEPA characters (<128) for money transfers (CCUTSEPA)
- characters valid in ISO-8859-15, CP1252 and IBM1142 (CCUTDELA)
- characters valid in ISO-8859-15, CP1252, IBM1142 and XOEF (CCUTDLAX)

All user table definitions are pre-calculated once at the beginning of execution to reduce the required CPU time for the conversion. So, the use of a user table increases the cost to open a file but has no effect on CPU utilization when converting the data.

To minimize the effort and to simplify the usage of user table, the most common subsets can be selected by keyword. In this case, a pre-calculated load module (DLL) is used. The subset definition can be for UTF (21 Bit UNICODE) or UCS (16 Bit UNICODE). If it is a UCS subset, then no valid code point is greater than 0xFFFF and you must use UCS CCSIDS to ensure that no code point greater than 0xFFFF is accepted.

If you have a working user table definition for a popular subset or system table manipulation, you can request a keyword for it. Please report this requirement over our issue tracking system

```
http://www.flam.de/en/technology/support/issues/
```

and attach the corresponding user table definition.

SELECTIONS

- `NPAS -UCS subset for statutory reporting (New passport, XOEF, String.Latin)`
- `SEPA -UCS subset of valid SEPA character smaller then 128`
- `DELA -UCS subset for German Latin of IBM1141, ISO8859-15 and CP1252`
- `DLAX -UCS subset for German Latin of IBM1141, ISO8859-15, CP1252 and XOEF`

4.5.46 PARAMETER *REPFIL*

SYNOPSIS

```
HELP:   Name of the report file to log substitutions
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   STRING
SYNTAX: REPFIL='str'/STDOUT/STDERR
```

DESCRIPTION With the report capability, you can log every substitution (stop / ignore / substitute / translit / skip / ...) to a log file. For the report file, all rules and replacements are valid, which are described under "FILENAME HANDLING" above. For example:

```
   REPORT=STDERR                          ; is SYSPRINT on z/OS
   REPORT='~.MYREPORT(REPORT1)'           ; PDS on z/OS
   REPORT='<SYSUID>.MYREPRT2'             ; PS dataset on z/OS
   REPORT='~/myreport3.txt'               ; Unix path name
   REPORT='<HOME>\reports\myreport4.txt'  ; Windows path name
```

A substitution report has the syntax listed below:

```
   report_file         -> sub_definition_list
   sub_definition_list -> sub_definition sub_definition_list
                       |  EMPTY
   sub_definition      -> SUB '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string '='
                             code_point_list byte_string ')'
                       -> XLT '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string '='
                             code_point_list byte_string ')'
                       |  IGN '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string ')'
                       |  STP '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string ')'
                       |  IDT '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string ')'
                       |  SKP '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string ')'
                       |  UKW '(' UNIT ':' OFFSET '+' POSITION ':'
                             code_point byte_string ')'
   code_point_list     -> CODEPOINT '/' code_point_list
                       |  EMPTY
   code_point          -> CODEPOINT
                       |  EMPTY
   byte_string         -> '(' BYTESTRING ')'
                       |  EMPTY
```

A comment behind brackets explains the reason for substitution:

```
'invalid character'
'incomplete character'
'incomplete character at the end of a record'
```

The notations STP (stop), IGN (ignore), SUB (substitute), XLT (transliteration), IDT (identical) or SKP (skip) before round brackets indicate the action taken. If the task can not be determined then UKW (unknown) is used. A ignore of an invalid character is realized as substitution with nothing. All substitutions with a length of 0 are indicated with IGN (ignore). The position of such an abnormal/irregular character is coded inside the braces:

- UNIT number of records or blocks (depends on the data)
- OFFSET byte offset up to this unit in the complete data stream

- POSITION number of bytes inside the unit where the activity was done

All counting starts with 0. The first block or record has unit 0.

The position is relative to the data stream which was presented to the character conversion module. After conversions and formatting the position values can be far off from the input file which was read originally. For example, when reading a text file with text formatting to records, then delimiting characters are not included in the byte offset, the units are records and the position indicates which character was the issue for this log entry.

The CODEPOINTs (UNICODE) are taken from the defined substitution table The BYTESTRINGs are taken from the input (read characters) and output data (written characters). BYTESTRINGs and CODEPOINTs are provided if they are known in the cases of such an irregularity. In some cases, these values can be missing.

For interpreting the report file, you must know that only the values inside round brackets are meaningful. Text before or behind round brackets are comments, but this may be subject to changes.

SELECTIONS

- `STDOUT -Write output to stdout`
- `STDERR -Write output to stderr`

4.5.47 PARAMETER *SKPBIN*

SYNOPSIS

```
HELP:   Skip character conversion if binary data is detected [FALSE]
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: SKPBIN
```

DESCRIPTION If set, this switch bypasses the character conversion module if the from-code cannot be detected. Charset detection is performed only when no CCSID or DEFAULT is provided. For automated conversion, it is useful to activate this mechanism. This may allow you to process binary and character data using the same command by performing character conversion only on printable, but not on binary data.

4.5.48 PARAMETER *SKPEQU*

SYNOPSIS

```
HELP:   Skip character conversion if from-code equals to-code [FALSE]
PATH:   XCHK.INPUT.SAV.FILE.CNV.CHR
TYPE:   SWITCH
SYNTAX: SKPEQU
```

DESCRIPTION If set, this switch bypasses the character conversion module if the from-code equals the to-code (which may happen when using charset auto detection), resulting in increased performance. This, however, also means that the input data will not be checked for invalid characters.

4.5.49 OBJECT *BAS*

SYNOPSIS

```
HELP:   Convert Base16/32/64/85 encoded data to binary data
PATH:   XCHK.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: BAS(BUFSIZ=num,RECCNT=num,METHOD=BASE16/BASE32/BASE64,MODE=BLOCK/RECORD,CHRSET=NONE ↩
    /SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/ ↩
    UTF32LE,CRCCHK,ARMCHK,RSTCHK,SKPNOB)
```

DESCRIPTION The base converter can be used to convert data from binary format to printable text and vice versa using one of a couple of base encoding schemes (RFC 4648). When writing, binary data is encoded, when reading, encoded text is decoded. Encoding data to a different base results in expansion of the data. The ratio of expansion depends on the base used.

Most base encoding schemes append some padding bytes to the end of output data to pad it to a multiple of a specific length. As a result, encoding and decoding can be performed in one of two modes:

The block mode converts the input data as one large block of binary data. Padding will only be added at the end of the output file. This mode is the default for any type of input data, unless specified otherwise. This implies that record-based data will not retain record boundaries after encoding. As the output is printable text when encoding, it can optionally be split into multiple lines. The *line* length parameter specifies after how many characters a line is wrapped. The line delimiter can be configured, but can also be omitted, which is useful to write the output record-oriented. When writing record-oriented, each output line results in one record. Note that if the *line* length parameter is specified and a line delimiter is set, then the total maximum record length will be line length plus the length of the delimiter.

The record mode can be useful if the input data is record-based. In record mode, each record is encoded independently, which results in retaining the ability to access specific records by only decoding those specific records. Each output record is padded as required by the respective encoding scheme. Due to base encoding resulting in expansion of the data, the resulting records will be longer after encoding and shrink when decoding. To read a file that has been encoded in record mode, setting mode=record is mandatory. Otherwise, the result might be garbage.

The character set used (ASCII or EBCDIC) can also be configured. By default, the system-specific character set is used.

Mainly for PGP support, the component supports optional ASCII armor (also in EBCDIC) headers and trailers around the base encoded data (see RFC4880), but can also be used for other data and is independent of the encoding method (Base16/32/64). ASCII-armored data data is wrapped into lines, which can be written as records or as blocks with lines and delimiters. The default line length is 72 if no length is specified. When reading, the trailer string can optionally be verified to match the header string.

Another feature for OpenPGP support is a CRC checksum that is appended to the data. This is also optional and usable independently of the data, the encoding schema and the ASCII armor support. When writing, a switch can be enabled to add a CRC24 checksum conforming to RCF4880 for Base64 encodings. The other encoding schemas use the same mechanism to append a checksum to the base encoded data, but use another CRC variant. Base16 uses CRC32 and Base32 uses CRC40, known from the GSM network. When reading, a switch enables verification of this checksum. By default, no CRC verification is done. Since the checksum is optional, it is useless to calculate the CRC just to find out that there is no CRC appended to the data that can be compared against. If the verification fails, you get an error. If the verification is successful, you get an informational. If there is no checksum appended for verification, you get a warning. The same is true if verification is not requested, but a checksum is found.

ARGUMENTS

- `NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]`
- `NUMBER:RECCNT=num -Initial amount of records for preallocation [128]`
- `NUMBER:METHOD=BASE16/BASE32/BASE64 -Base encoding selector [AUTO]`
  - `BASE16 -Base16 (hexadecimal) encoding`
  - `BASE32 -Base32 encoding`
  - `BASE64 -Base64 encoding`
- `NUMBER:MODE=BLOCK/RECORD -Enforces record or block-oriented conversion [BLOCK]`
  - `BLOCK -Block-oriented conversion (all data is one chunk)`
  - `RECORD -Record-oriented conversion (convert each record individually)`
- `NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [SYSTEM]`
  - `NONE -No character set defined`
  - `SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)`

  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- SWITCH:CRCCHK -Check CRC if available [FALSE]
- SWITCH:ARMCHK -Check ARMOR trailer if available [FALSE]
- SWITCH:RSTCHK -Check if a rest after encoded data [FALSE]
- SWITCH:SKPNOB -Skip base decoding if no base encoding detected [FALSE]

4.5.50 OBJECT *HSH*

SYNOPSIS

```
HELP:   Generation or verification of one way hash values
PATH:   XCHK.INPUT.SAV.FILE.CNV
TYPE:   OBJECT
SYNTAX: HSH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/ ↩
    CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)

  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.5.51 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   XCHK.INPUT.SAV.FILE.CNV.HSH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.5.52 OVERLAY *FMT*

SYNOPSIS

```
HELP:   Formatting [BIN]
PATH:   XCHK.INPUT.SAV.FILE
TYPE:   OVERLAY
SYNTAX: FMT.{BIN()/BLK()/REC()/TXT()/XML()}
```

DESCRIPTION With the overlay FMT you can choose the different kind of formatting methods to convert the data lists in elements at read operations.

4.5.53 OBJECT *BIN*

SYNOPSIS

```
HELP:   Format data in binary elements (block/records)
PATH:   XCHK.INPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: BIN(BUFSIZ=num,RECCNT=num,TERASE,SEGINT=NONE/CHKS/AUTH/MAC,LENMOD=NONE/FAST/DYNAMIC ↩
    /COMPACT,LENCNF=NONE/VEIL/ENC,DATMOD=NONE/FAST/DYNAMIC/COMPACT,DATCNF=NONE/VEIL/ENC)
```

DESCRIPTION The object "format binary" converts each item from a data list (blocks or records with attributes) to an element. The data is untouched, the assigned element's types depend on the attributes and the attributes are stored for each element. This formatting method is mainly useful for records or binary chunks of data.

ARGUMENTS

- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:RECCNT=num -Initial amount of records for preallocation [128]
- SWITCH:TERASE -Erase data from memory when no longer needed [FALSE]
- NUMBER:SEGINT=NONE/CHKS/AUTH/MAC -Segment integrity mode [NONE]
  - NONE -No integrity protection
  - CHKS -Integrity protection with checksum
  - AUTH -Integrity protection with encrypted checksum
  - MAC -Integrity protection with message authentication codes
- NUMBER:LENMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for length fields [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:LENCNF=NONE/VEIL/ENC -Encryption mode for length fields [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption
- NUMBER:DATMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for data [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)

  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:DATCNF=NONE/VEIL/ENC -Encryption mode for data [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption

4.5.54 OBJECT *BLK*

SYNOPSIS

```
HELP:   Format data in one block element (all meta data lost)
PATH:   XCHK.INPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: BLK(BUFSIZ=num,TERASE,SEGINT=NONE/CHKS/AUTH/MAC,LENMOD=NONE/FAST/DYNAMIC/COMPACT, ↩
    LENCNF=NONE/VEIL/ENC,DATMOD=NONE/FAST/DYNAMIC/COMPACT,DATCNF=NONE/VEIL/ENC)
```

DESCRIPTION The object "format block" formats all data items (a block or a bunch of records) as an element containing the raw block/chunk of data. All additional attributes and record boundaries are lost.

ARGUMENTS

- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- SWITCH:TERASE -Erase data from memory when no longer needed [FALSE]
- NUMBER:SEGINT=NONE/CHKS/AUTH/MAC -Segment integrity mode [NONE]
  - NONE -No integrity protection
  - CHKS -Integrity protection with checksum
  - AUTH -Integrity protection with encrypted checksum
  - MAC -Integrity protection with message authentication codes
- NUMBER:LENMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for length fields [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:LENCNF=NONE/VEIL/ENC -Encryption mode for length fields [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption
- NUMBER:DATMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for data [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:DATCNF=NONE/VEIL/ENC -Encryption mode for data [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption

4.5.55 OBJECT *REC*

SYNOPSIS

```
HELP:   Format data in several wrapped record elements (all meta data lost)
PATH:   XCHK.INPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: REC(METHOD=L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX,WRPERR,RECLEN=num,BUFSIZ ↩
    =num,INICNT=num,TERASE,SEGINT=NONE/CHKS/AUTH/MAC,LENMOD=NONE/FAST/DYNAMIC/COMPACT,LENCNF ↩
    =NONE/VEIL/ENC,DATMOD=NONE/FAST/DYNAMIC/COMPACT,DATCNF=NONE/VEIL/ENC)
```

DESCRIPTION By default (i.e. no method provided), the object "format record" formats data blocks to records by slicing the block into records of the provided record length if none of the valid 4 byte length formats is detected.If the data is already organized as records, then the data is re-wrapped. You can enforce an error to prevent the default wrapping if no record length information is found.

ARGUMENTS

- NUMBER:METHOD=L4I/L4X/B4I/B4X/S4I/S4X/HLI/HLX/HBI/HBX/HSI/HSX -Method for record for matting [AUTO]
  - L4I -Parse data based on 4 byte length fields:Little endian integer, length inclusive
  - L4X -Parse data based on 4 byte length fields:Little endian integer, length exclusive (ZIP)
  - B4I -Parse data based on 4 byte length fields:Big endian integer, length inclusive
  - B4X -Parse data based on 4 byte length fields:Big endian integer, length exclusive (USS)
  - S4I -Parse data based on 4 byte length fields:System endian integer, length inclusive
  - S4X -Parse data based on 4 byte length fields:System endian integer, length exclusive (VAR)
  - HLI -Parse data based on 4 byte length fields:Little endian short (LLxx), length inclusive
  - HLX -Parse data based on 4 byte length fields:Little endian short (LLxx), length exclusive
  - HBI -Parse data based on 4 byte length fields:Big endian short (LLxx), length inclusive (MVS)
  - HBX -Parse data based on 4 byte length fields:Big endian short (LLxx), length exclusive
  - HSI -Parse data based on 4 byte length fields:System endian short (LLxx), length inclusive
  - HSX -Parse data based on 4 byte length fields:System endian short (LLxx), length exclusive
- SWITCH:WRPERR -If no method defined (auto detection) enforce an error if no length format (wrapping) detected [FALSE]
- NUMBER:RECLEN=num -Maximum length of a record [512]
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of records for preallocation [128]
- SWITCH:TERASE -Erase data from memory when no longer needed [FALSE]
- NUMBER:SEGINT=NONE/CHKS/AUTH/MAC -Segment integrity mode [NONE]
  - NONE -No integrity protection

  - CHKS -Integrity protection with checksum
  - AUTH -Integrity protection with encrypted checksum
  - MAC -Integrity protection with message authentication codes
- NUMBER:LENMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for length fields [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:LENCNF=NONE/VEIL/ENC -Encryption mode for length fields [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption
- NUMBER:DATMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for data [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:DATCNF=NONE/VEIL/ENC -Encryption mode for data [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption

4.5.56 OBJECT *TXT*

SYNOPSIS

```
HELP:   Format data based on delimiters in text elements (record and rest)
PATH:   XCHK.INPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: TXT(METHOD=BIN/DLM/REC,CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/ ↩
    UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE,RPLFFD=num,SUPTWS,NELDLM,PADCHR=num,RECLEN= ↩
    num,BUFSIZ=num,INICNT=num,BINERR,CHRERR,TERASE,SEGINT=NONE/CHKS/AUTH/MAC,LENMOD=NONE/ ↩
    FAST/DYNAMIC/COMPACT,LENCNF=NONE/VEIL/ENC,DATMOD=NONE/FAST/DYNAMIC/COMPACT,DATCNF=NONE/ ↩
    VEIL/ENC)
```

DESCRIPTION The object "format text" splits the data into text records and rest elements based on delimiters (METHOD=DLM). The rest element contains the delimiter and optional trailing whitespace if the suppression of trailing whitespace is activated. The text record and the rest element together compose the original data.

If the data contains no delimiters but records length are available, record formatting (METHOD=REC) is used. In this case, the rest element is empty unless suppression of trailing whitespace is used. If no delimiters and no length fields are found and the BINERR switch is not set, then binary formatting (wrapping of the data into records of the defined length) is done. If the BINERR switch is set, an error is triggered if the data is binary. Delimiter, record or binary formatting can be enforced by defining the corresponding method or the data dependent automatism can be used.

If you have text in one of the ISO code pages with 0x85 (NEL) as delimiter, you can optionally enable recognition of NEL as delimiter. NEL (next line) is optional because this value is the general currency sign or the continuation symbol and not a new line in many single byte ASCII codepages.

- 0x00 (padding character (can be changed))
- 0x0A (carriage return (EBCDIC 0x25))
- 0x0D (line feed (EBCDIC(0x0D)))
- 0x0A, 0x0D (carriage return line feed)
- 0x0A, . . . , 0x0A, 0x0D (dirty delimiters)
- 0x0A, 0x0D, . . . , 0x0D (dirty delimiters)
- 0x0C (form feed (EBCDIC 0x22))
- 0x85 (new/next line (EBCDIC 0x15, optional if single byte ASCII))

Text formatting supports a lot of powerful features which can be accessed with the parameters below.

ARGUMENTS

- NUMBER:METHOD=BIN/DLM/REC -Method for text formatting [DEFAULT]
  - BIN -Wrap data block in records of record length
  - DLM -Parse data block for text delimiter (DEFAULT)
  - REC -Parse data block based on record length fields
- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [auto detection]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assuming n lines per page [0 =no replacement]
- SWITCH:SUPTWS -Suppress trailing whitespaces [FALSE]
- SWITCH:NELDLM -Activate NEL (0x85) as delimiter for ASCII character sets [FALSE]
- NUMBER:PADCHR=num -Padding character (is regarded as additional delimiter) [0x00]
- NUMBER:RECLEN=num -Maximum length of a text record (Host) /line (Unix/Win) [512]
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of records for preallocation [128]

- SWITCH:BINERR –Enforce an error if binary data detected [FALSE]
- SWITCH:CHRERR –Enforce an error if character data without delimiter detected [FALSE]
- SWITCH:TERASE –Erase data from memory when no longer needed [FALSE]
- NUMBER:SEGINT=NONE/CHKS/AUTH/MAC –Segment integrity mode [NONE]
  - NONE –No integrity protection
  - CHKS –Integrity protection with checksum
  - AUTH –Integrity protection with encrypted checksum
  - MAC –Integrity protection with message authentication codes
- NUMBER:LENMOD=NONE/FAST/DYNAMIC/COMPACT –Compression mode for length fields [COPY]
  - NONE –No compression =copy of data
  - FAST –Fast compression (more data, less CPU)
  - DYNAMIC –Dynamic compression (best in data and CPU)
  - COMPACT –Compact compression (less data, more CPU)
- NUMBER:LENCNF=NONE/VEIL/ENC –Encryption mode for length fields [NONE]
  - NONE –No confidentiality protection
  - VEIL –Confidentiality protection with veiling
  - ENC –Confidentiality protection with encryption
- NUMBER:DATMOD=NONE/FAST/DYNAMIC/COMPACT –Compression mode for data [COPY]
  - NONE –No compression =copy of data
  - FAST –Fast compression (more data, less CPU)
  - DYNAMIC –Dynamic compression (best in data and CPU)
  - COMPACT –Compact compression (less data, more CPU)
- NUMBER:DATCNF=NONE/VEIL/ENC –Encryption mode for data [NONE]
  - NONE –No confidentiality protection
  - VEIL –Confidentiality protection with veiling
  - ENC –Confidentiality protection with encryption

4.5.57 OBJECT *XML*

SYNOPSIS

```
HELP:   Format data stream in XML elements (tags, attributes, data, ...)
PATH:   XCHK.INPUT.SAV.FILE.FMT
TYPE:   OBJECT
SYNTAX: XML(CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/ ↩
    UTF32BE/UCS4LE/UTF32LE,FMTERR,NOCMNT,BUFSIZ=num,INICNT=num,TERASE,SEGINT=NONE/CHKS/AUTH/ ↩
    MAC,LENMOD=NONE/FAST/DYNAMIC/COMPACT,LENCNF=NONE/VEIL/ENC,DATMOD=NONE/FAST/DYNAMIC/ ↩
    COMPACT,DATCNF=NONE/VEIL/ENC)
```

DESCRIPTION The object "format XML" parses block of text data containing an XML document. The XML data is parsed using the Expat library and transformed into FLAM elements, which allows various formatting options when writing, including minimizing and pretty printing the XML data.

The text data must be in UTF-8 or ASCII. Lines must be delimited with 0x0A, 0x0D or 0x0D0A. During parsing, all line delimiters are normalized to line feed characters (0xA) as defined by the XML specification.

XML formatting supports a lot of powerful features which can be accessed with the parameters below.

ARGUMENTS

- NUMBER:CHRSET=NONE/SYSTEM/ASCII/UCS1/UTF8/EBCDIC/UCS2BE/UTF16BE/UCS2LE/UTF16LE/UCS4BE/UTF32BE/UCS4LE/UTF32LE -Character set [auto]
  - NONE -No character set defined
  - SYSTEM -SYSTEM (on IBM mainframe EBCDIC else ASCII)
  - ASCII -ASCII (mainly used in the for open system)
  - UCS1 -UCS-1 (for text formatting identical to ASCII < 64k)
  - UTF8 -UTF-8 (for text formatting identical to ASCII < 2M)
  - EBCDIC -EBCDIC (mainly used on IBM mainframe)
  - UCS2BE -UCS-2 Big Endian (multibyte character set < 64k)
  - UTF16BE -UTF-16 Big Endian (multibyte character set < 2M)
  - UCS2LE -UCS-2 Little Endian (multibyte character set < 64k)
  - UTF16LE -UTF-16 Little Endian (multibyte character set < 2M)
  - UCS4BE -UCS-4 Big Endian (multibyte character set < 64k)
  - UTF32BE -UTF-32 Big Endian (multibyte character set < 2M)
  - UCS4LE -UCS-4 Little Endian (multibyte character set < 64k)
  - UTF32LE -UTF-32 Little Endian (multibyte character set < 2M)
- SWITCH:FMTERR -Enforce a format error if data contains only space or is empty [FALSE]
- SWITCH:NOCMNT -Ignore XML comments [FALSE]
- NUMBER:BUFSIZ=num -Initial buffer size for preallocation [65536]
- NUMBER:INICNT=num -Initial amount of records for preallocation [128]
- SWITCH:TERASE -Erase data from memory when no longer needed [FALSE]
- NUMBER:SEGINT=NONE/CHKS/AUTH/MAC -Segment integrity mode [NONE]
  - NONE -No integrity protection
  - CHKS -Integrity protection with checksum
  - AUTH -Integrity protection with encrypted checksum
  - MAC -Integrity protection with message authentication codes
- NUMBER:LENMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for length fields [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:LENCNF=NONE/VEIL/ENC -Encryption mode for length fields [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption
- NUMBER:DATMOD=NONE/FAST/DYNAMIC/COMPACT -Compression mode for data [COPY]
  - NONE -No compression =copy of data
  - FAST -Fast compression (more data, less CPU)
  - DYNAMIC -Dynamic compression (best in data and CPU)
  - COMPACT -Compact compression (less data, more CPU)
- NUMBER:DATCNF=NONE/VEIL/ENC -Encryption mode for data [NONE]
  - NONE -No confidentiality protection
  - VEIL -Confidentiality protection with veiling
  - ENC -Confidentiality protection with encryption

4.5.58 OBJECT *ENV*

SYNOPSIS

```
HELP:   Environment
PATH:   XCHK.INPUT.SAV.FILE
TYPE:   OBJECT
SYNTAX: ENV[(KEYWORD='str',VALUE='str')...]
```

DESCRIPTION With the object ENV you can overrule up to 32 environment variables for read operation. This is useful to define (for example) another LANG variable to emulate another platform for character conversion.

ARGUMENTS

- `STRING:KEYWORD='str' -Keyword of environment variable`
- `STRING:VALUE='str' -Value of environment variable`

4.5.59 OBJECT *DIR*

SYNOPSIS

```
HELP:   Defines how to handle directories and various file types if wildcards used
PATH:   XCHK
TYPE:   OBJECT
SYNTAX: DIR(LINK,ALIAS,HIDDEN,RECURSIVE,ARCHIVE,ONEFILESYSTEM,TAPE)
```

DESCRIPTION This object defines multiple switches that control how input files are found. This is especially important when using wildcards in input filenames.

There are switches to enable recursion into subdirectories, following symbolic links, finding hidden files, recursion into archive files, include files on tapes, resolving aliases and enforcing to not leave the filesystem.

If the recursive flag is set, the filename portion of the input file string is searched for recursively. If a directory path containing wildcards is present, the filename portion is searched for recursively in all paths that match the directory path pattern, e.g. "**/path/**.txt" matches my/path/hugo.txt, another/path/hugo.txt as well as my/path/subdir/hugo.txt. Without the recursive flag, the last file would not be found.

If the archive switch is enabled, FLAMFILES, TARBALLS, ZIP files and other kinds of supported archives are searched. All the matching members are extracted. In other words, archives are interpreted like subdirectories if the flag is enabled.

If the tape switch is enabled, then datasets stored on tapes are searched. If the name of such a file matches the pattern, then it might occur that console input and a manual intervention is required to retrieve the respective dataset(s).

If the link switch is enabled, symbolic links are followed (otherwise they are ignored). Links are mainly supported on modern file systems. For dataset aliases on mainframe systems, see the alias flag.

With the alias switch enabled, datasets names which are aliases of the real dataset are resolved to their actual names (otherwise they are ignored). Dataset aliases only exist on mainframe systems. This flag has no effect on other platforms.

If a transparent read method (e.g. read.auto()) is used, then the archive flag is enabled by default and cannot be disabled.

ARGUMENTS

- `SWITCH:LINK -Include symbolic links`
- `SWITCH:ALIAS -Resolve data set aliases`
- `SWITCH:HIDDEN -Include hidden files`
- `SWITCH:RECURSIVE -Include sub directories`
- `SWITCH:ARCHIVE -Include archives (FLAMFILEs, ZIP, ...)`
- `SWITCH:ONEFILESYSTEM -Don't cross file system boundaries`
- `SWITCH:TAPE -Include files deposited on tapes`

4.5.60 OBJECT *LOG*

SYNOPSIS

```
HELP:   Logging Parameter
PATH:   XCHK
TYPE:   OBJECT
SYNTAX: LOG(MESSAGE(),DESTINATION.{})
```

DESCRIPTION The LOG parameter allows flexible adjustment for the logging of messages. It is possible to specify the destination as well as which kind of message will be written.

For more information please have a look at the man pages for each parameter of LOG.

4.5.61 OBJECT *MESSAGE*

SYNOPSIS

```
HELP:   Select type of messages to log
PATH:   XCHK.LOG
TYPE:   OBJECT
SYNTAX: MESSAGE(NONE,ERROR,ERRTRACE,WARNING,NOTICE,DEBUG,ABOUT,VERSION,INPUT,PARAMETER, ↩
    STATISTIC,INFO,LICID,LICENSE,PROGRESS,ALL,DEFAULT,MINIMAL,ERRONLY)
```

DESCRIPTION The log message specification is a classification of the messages to write to the selected log destination. Each type is selectable individually by adding its argument inside this object.

For added convenience, the special types NON, ALL, MINIMAL select the proper message types with just one argument.

If no message specification is given, a DEFAULT is used which writes out messages of these types:

- ERROR
- ERRTRACE
- WARNING
- NOTICE
- PARAMETER
- STATISTIC
- INFO
- LICID

ARGUMENTS

- `SWITCH:NONE -No messages will be logged`
- `SWITCH:ERROR -Error messages will be logged`
- `SWITCH:ERRTRACE -Error trace will be logged`
- `SWITCH:WARNING -Warnings will be logged`
- `SWITCH:NOTICE -Notices will be logged`
- `SWITCH:DEBUG -Debug information will be logged`
- `SWITCH:ABOUT -About information will be logged`

- SWITCH:VERSION -Version information will be logged
- SWITCH:INPUT -Input data will be logged (dangerous for passwords)
- SWITCH:PARAMETER -Parsed parameters will be logged
- SWITCH:STATISTIC -Statistics will be logged
- SWITCH:INFO -Requested information will be logged
- SWITCH:LICID -License ID will be logged
- SWITCH:LICENSE -License summary will be logged
- SWITCH:PROGRESS -Progress bar will be logged
- SWITCH:ALL -All messages are logged
- SWITCH:DEFAULT -Relevant messages are logged
- SWITCH:MINIMAL -Only important messages are logged
- SWITCH:ERRONLY -Only errors and error trace are logged

4.5.62 OVERLAY *DESTINATION*

SYNOPSIS

```
HELP:   Destination for log messages
PATH:   XCHK.LOG
TYPE:   OVERLAY
SYNTAX: DESTINATION.{STREAM()/FILE()/SYSTEM()}
```

DESCRIPTION The log destination parameter defines where the log messages will be written.

Each log message starts with the current time and date.

All the possible destinations allow to specify a custom identification string which will be printed after time and date of each message. This is done with the IDENT argument.

If no IDENT specified the default will be *owner.program.command*

If no log destination defined stream will be used as default.

4.5.63 OBJECT *STREAM*

SYNOPSIS

```
HELP:   Log to a stream
PATH:   XCHK.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: STREAM(FORMAT=STANDARD/DIALOG,IDENT='str',STDOUT)
```

DESCRIPTION The STREAM object allows the log messages to be written to the standard output or the standard error output. Standard error output is used if STDOUT is not specified.

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- SWITCH:STDOUT -Log messages to STDOUT instead of STDERR

4.5.64 OBJECT *FILE*

SYNOPSIS

```
HELP:   Log to a flat file
PATH:   XCHK.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: FILE(FORMAT=STANDARD/DIALOG,IDENT='str',NAME='str')
```

DESCRIPTION The flat file option allows the log messages to be written to a normal file. The name of the file must be specified for this destination. The log messages will be appended to this file if it already exists.

For the log file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   NAME='~.MYLOG(LOG1)'                    ; PDS on z/OS
   NAME='<SYSUID>.MYLOG02'                 ; PS dataset on z/OS
   NAME='~/mylog3.txt'                     ; Unix path name
   NAME='<HOME>\logs\mylog4.txt'           ; Windows path name
```

ATTENTION: Parallel process cannot share the same log file

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- STRING:NAME='str' -Filename for appending log messages

4.5.65 OBJECT *SYSTEM*

SYNOPSIS

```
HELP:   Use system logging facility
PATH:   XCHK.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: SYSTEM(IDENT='str')
```

DESCRIPTION The SYSTEM destination allows the log messages to be written to the logging system of the operating system.

On Unix and z/OS® systems the syslog() call is used. On Windows® the event logging facility is used.

```
NOTE: On Windows(R) this requires the registry key entry:
      HKEY_LOCAL_MACHINE
        SYSTEM
          CurrentControlSet
            Services
              EventLog
                Application
                  FLM
```

This registry key will be created by using the DLL flevent.dll

ARGUMENTS

- STRING:IDENT='str' -Identifier used to identify log entries

4.6 COMMAND *HASH*

SYNOPSIS

```
HELP:   Hash calculation/verification
PATH:   LIM.flcl
TYPE:   OBJECT
SYNTAX: :> flcl HASH(FILE='str'/STREAM/DUMMY,DIR(),LOG(),ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/ ↩
    SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT= ↩
    GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The HASH command calculates or verifies checksums for one or more input files. Files are read in binary format. Checksums in GNU and BSD formats are supported, which are compatible with common tools like *sha1sum* and *md5sum* on UNIX systems. This can be useful to verify files transferred between different systems, especially to or from mainframes.

If possible, hardware acceleration is used for hash calculation.

The HASH command prints only minimal log output. For a more verbose output (e.g. statistics), please use the log object (see examples below).

To get syntax information, please use:

```
   flcl SYNTAX HASH
```

To get help for the parameters, please use:

```
   flcl HELP HASH.parameter[.parameter[...]]
```

To read the manual page for a parameter, please use:

```
   flcl MANPAGE HASH.parameter[.parameter[...]]
      or
   flcl HELP XCHK.parameter[.parameter[...]] MAN
```

To generate the user manual for the command, please use:

```
   flcl GENDOCU HASH=filename
```

Parameters can be defined via command line (directly or per file) or through properties read from the corresponding property file.

EXAMPLES Calculate the checksum using the default hash algorithm (SHA1) for the file "fileToBeHashed.exe" in default format (GNU) to the default output file (LOG).

```
   flcl hash file=fileToBeHashed.exe
```

Calculate the checksum using SHA-512 hash algorithm for the file "fileToBeHashed.exe".

```
   flcl hash file=fileToBeHashed.exe algo=sha512
```

Calculate the checksum using SHA-512 hash algorithm for the file "fileToBeHashed.exe" and write the result to the file "outhash.txt".

```
   flcl hash file=fileToBeHashed.exe algo=sha512 output=outhash.txt
```

Verify the file "fileToBeHashed.exe" using SHA-512 hash algorithm by comparing to the checksum stored in file "outhash.txt".

```
   flcl hash file=fileToBeHashed.exe algo=sha512 check.file=outhash.txt
```

Calculate the checksum using SHA-512 hash algorithm for the file "fileToBeHashed.exe". Print log output with the default filter (more output information)

```
   flcl hash "file=fileToBeHashed.exe algo=sha512 log(message(default))"
```

If you want to get the hash value of the decompressed content of a file without decompressing it to a temporary file, please take a look at the XCHK command. It is a much more powerful command to analyze input files.

ARGUMENTS

- STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to hash ["==origin.ext]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.6.1 OBJECT *DIR*

SYNOPSIS

```
HELP:   Defines how to handle directories and various file types if wildcards used
PATH:   HASH
TYPE:   OBJECT
SYNTAX: DIR(LINK,ALIAS,HIDDEN,RECURSIVE,ARCHIVE,ONEFILESYSTEM,TAPE)
```

DESCRIPTION This object defines multiple switches that control how input files are found. This is especially important when using wildcards in input filenames.

There are switches to enable recursion into subdirectories, following symbolic links, finding hidden files, recursion into archive files, include files on tapes, resolving aliases and enforcing to not leave the filesystem.

If the recursive flag is set, the filename portion of the input file string is searched for recursively. If a directory path containing wildcards is present, the filename portion is searched for recursively in all paths that match the directory path pattern, e.g. "**/path/**.txt" matches my/path/hugo.txt, another/path/hugo.txt as well as my/path/subdir/hugo.txt. Without the recursive flag, the last file would not be found.

If the archive switch is enabled, FLAMFILES, TARBALLS, ZIP files and other kinds of supported archives are searched. All the matching members are extracted. In other words, archives are interpreted like subdirectories if the flag is enabled.

If the tape switch is enabled, then datasets stored on tapes are searched. If the name of such a file matches the pattern, then it might occur that console input and a manual intervention is required to retrieve the respective dataset(s).

If the link switch is enabled, symbolic links are followed (otherwise they are ignored). Links are mainly supported on modern file systems. For dataset aliases on mainframe systems, see the alias flag.

With the alias switch enabled, datasets names which are aliases of the real dataset are resolved to their actual names (otherwise they are ignored). Dataset aliases only exist on mainframe systems. This flag has no effect on other platforms.

If a transparent read method (e.g. read.auto()) is used, then the archive flag is enabled by default and cannot be disabled.

ARGUMENTS

- `SWITCH:LINK -Include symbolic links`
- `SWITCH:ALIAS -Resolve data set aliases`
- `SWITCH:HIDDEN -Include hidden files`
- `SWITCH:RECURSIVE -Include sub directories`
- `SWITCH:ARCHIVE -Include archives (FLAMFILEs, ZIP, ...)`
- `SWITCH:ONEFILESYSTEM -Don't cross file system boundaries`
- `SWITCH:TAPE -Include files deposited on tapes`

4.6.2 OBJECT *LOG*

SYNOPSIS

```
HELP:   Logging Parameter
PATH:   HASH
TYPE:   OBJECT
SYNTAX: LOG(MESSAGE(),DESTINATION.{})
```

DESCRIPTION The LOG parameter allows flexible adjustment for the logging of messages. It is possible to specify the destination as well as which kind of message will be written.

For more information please have a look at the man pages for each parameter of LOG.

4.6.3 OBJECT *MESSAGE*

SYNOPSIS

```
HELP:   Select type of messages to log
PATH:   HASH.LOG
TYPE:   OBJECT
SYNTAX: MESSAGE(NONE,ERROR,ERRTRACE,WARNING,NOTICE,DEBUG,ABOUT,VERSION,INPUT,PARAMETER, ↩
    STATISTIC,INFO,LICID,LICENSE,PROGRESS,ALL,DEFAULT,MINIMAL,ERRONLY)
```

DESCRIPTION The log message specification is a classification of the messages to write to the selected log destination. Each type is selectable individually by adding its argument inside this object.

For added convenience, the special types NON, ALL, MINIMAL select the proper message types with just one argument.

If no message specification is given, a DEFAULT is used which writes out messages of these types:

- ERROR
- ERRTRACE
- WARNING
- NOTICE
- PARAMETER
- STATISTIC
- INFO
- LICID

ARGUMENTS

- SWITCH:NONE –No messages will be logged
- SWITCH:ERROR –Error messages will be logged
- SWITCH:ERRTRACE –Error trace will be logged
- SWITCH:WARNING –Warnings will be logged
- SWITCH:NOTICE –Notices will be logged
- SWITCH:DEBUG –Debug information will be logged
- SWITCH:ABOUT –About information will be logged
- SWITCH:VERSION –Version information will be logged
- SWITCH:INPUT –Input data will be logged (dangerous for passwords)
- SWITCH:PARAMETER –Parsed parameters will be logged
- SWITCH:STATISTIC –Statistics will be logged
- SWITCH:INFO –Requested information will be logged
- SWITCH:LICID –License ID will be logged
- SWITCH:LICENSE –License summary will be logged
- SWITCH:PROGRESS –Progress bar will be logged
- SWITCH:ALL –All messages are logged
- SWITCH:DEFAULT –Relevant messages are logged
- SWITCH:MINIMAL –Only important messages are logged
- SWITCH:ERRONLY –Only errors and error trace are logged

4.6.4 OVERLAY *DESTINATION*

SYNOPSIS

```
HELP:   Destination for log messages
PATH:   HASH.LOG
TYPE:   OVERLAY
SYNTAX: DESTINATION.{STREAM()/FILE()/SYSTEM()}
```

DESCRIPTION The log destination parameter defines where the log messages will be written.

Each log message starts with the current time and date.

All the possible destinations allow to specify a custom identification string which will be printed after time and date of each message. This is done with the IDENT argument.

If no IDENT specified the default will be *owner.program.command*

If no log destination defined stream will be used as default.

4.6.5 OBJECT *STREAM*

SYNOPSIS

```
HELP:   Log to a stream
PATH:   HASH.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: STREAM(FORMAT=STANDARD/DIALOG,IDENT='str',STDOUT)
```

DESCRIPTION The STREAM object allows the log messages to be written to the standard output or the standard error output. Standard error output is used if STDOUT is not specified.

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- SWITCH:STDOUT -Log messages to STDOUT instead of STDERR

4.6.6 OBJECT *FILE*

SYNOPSIS

```
HELP:   Log to a flat file
PATH:   HASH.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: FILE(FORMAT=STANDARD/DIALOG,IDENT='str',NAME='str')
```

DESCRIPTION The flat file option allows the log messages to be written to a normal file. The name of the file must be specified for this destination. The log messages will be appended to this file if it already exists.

For the log file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   NAME='~.MYLOG(LOG1)'                  ; PDS on z/OS
   NAME='<SYSUID>.MYLOG02'               ; PS dataset on z/OS
   NAME='~/mylog3.txt'                   ; Unix path name
   NAME='<HOME>\logs\mylog4.txt'         ; Windows path name
```

ATTENTION: Parallel process cannot share the same log file

ARGUMENTS

- `NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]`
  - `STANDARD -Standard logging format (timestamp ident *level* message)`
  - `DIALOG -For dialog processing (ident *level* message)`
- `STRING:IDENT='str' -Identifier used to identify log entries`
- `STRING:NAME='str' -Filename for appending log messages`

4.6.7 OBJECT *SYSTEM*

SYNOPSIS

```
HELP:   Use system logging facility
PATH:   HASH.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: SYSTEM(IDENT='str')
```

DESCRIPTION The SYSTEM destination allows the log messages to be written to the logging system of the operating system.

On Unix and z/OS® systems the syslog() call is used. On Windows® the event logging facility is used.

```
NOTE: On Windows(R) this requires the registry key entry:
      HKEY_LOCAL_MACHINE
        SYSTEM
          CurrentControlSet
            Services
              EventLog
                Application
                  FLM
```

This registry key will be created by using the DLL flevent.dll

ARGUMENTS

- `STRING:IDENT='str' -Identifier used to identify log entries`

4.6.8 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7 COMMAND *DIFF*

SYNOPSIS

```
HELP:   Compares two data sources
PATH:   LIM.flcl
TYPE:   OBJECT
SYNTAX: :> flcl DIFF(READ.{},COMPARE.{},LOGGING.{},MESSAGE(),FULLY,OFFSET,REPORT='str'/ ↩
    STDOUT/STDERR)
```

DESCRIPTION The DIFF command uses all reading capabilities of the CONV command to compare the clear net content of two data sources independent of the data format and representation. The comparison takes place on a logical level, based on the neutral FLAM5 elements.

The DIFF function does not recognize insertions or deletions to/from either source. If is one byte or element more or less in the data of one of the sources, then the comparison will fail for the whole rest.

This command does not support wildcards.

As an example, the DIFF command can be used to compare the logical content of two XML files, where one file is in UTF-16LE and GZIP compressed and the other one is a host data set compressed, encrypted and in EBCDIC as FLAMFILE member. The DIFF command builds a neutral element list (UTF-8) and compares the parsed XML elements.

The following features are supported:

- Different kinds of remote access protocols (IP-Native, MQ-Series, SSH)
- Different kinds of data sources (Files, Streams, Tables)
- Different kinds of archive formats (ZIP, FLAMFILE, TAR)
- Different kinds of I/O methods (Binary, Character, Text, Record, FLAM)
- Different data encodings (HEX (Base16), Base32, Base64)
- Different encryption methods (FLAM, PGP)
- Different compression methods (GZIP, BZIP2, XZ)
- Automatic decoding of data encodings during read operations
- Auto-detection and decoding of length fields in data blocks
- Auto-detection of character sets and language based conversion

- Powerful character conversion based on CCSIDs with subset support and reporting
- Different logical data formatting (BIN, TEXT, XML, . . .)

Depending on the data formats, a dedicated method is used, to compare the logical data records or streams. If a compare of the data formats makes no sense (XML with records or blocks) then an error is issued. For example: If you use read.file=*filename* and compare.file=*filename* and the first file is binary and the second file is the compressed version of this binary data, then a list of binary blocks with different length is built. In this case, it makes no sense to compare the elements (they are several blocks of different sizes). Hence, we compare the logical data stream and not the element list. If the first file contains records from a host data set and the second file contains text records with trailing white spaces and line delimiter (=rest element), then we compare only the net record elements. If the first file is block- oriented, then we include the remaining rest element from the text parser in a data stream compare.

With the switch *FULLY* you can activate additional checks depending on the data formats, for example:

- TEXT/TEXT: Compare the remaining rest (trailing whitespace and delimiter)
- ASA-RECORD/ASA-RECORD: Compare the control character
- REL-RECORD/REL-RECORD: Compare the slot numbers
- XML/XML: Compare comments, formatting (whitespace) and DTD

If you want to verify that two files are identical independent of the platform and representation, then don't use this switch.

If you specified a report file and activate the *OFFSET* switch, then byte offsets and byte values for each difference are written to the report file. ATTENTION: This can cause a very large report file!

To get syntax information, please use:

```
flcl SYNTAX DIFF
```

To get help for a parameter, please use:

```
flcl HELP DIFF.parameter[.parameter[...]]
```

To read the manual page for a parameter, please use:

```
flcl MANPAGE DIFF.parameter[.parameter[...]]
   or
flcl HELP DIFF.parameter[.parameter[...]] MAN
```

To generate the user manual for the command, please use:

```
flcl GENDOCU DIFF=filename
```

Parameters can be defined via command line (directly or by parameter file) or via properties taken from the corresponding property file

EXAMPLE

```
flcl DIFF read.file='./swift.txt.gz' compare.file='~.swift.fba' report='out.txt'
```

This test reads text records and remaining rest elements from a text file which was compressed and for example encoded in ASCII-CP1252 and contains 0x0D0A as record delimiter. On the other side, an FBA host data set in EBCDIC is read and the ASA control characters are detached as attributes and trailing spaces are removed. Both files are converted to UTF-8 and only the net record elements are compared.

```
flcl DIFF read.xml(file='test.xml.bz')
            compare.xml(file='test.xml')
            report='out.txt' fully offset
```

This example requires and compares two XML files including comments, DTD and whitespace element by element. The report file contains the byte offsets for each difference. If you don't use the *FULLY* switch, then a data stream compare is done, i.e. only tags, attributes and data elements are compared.

```
   flcl DIFF read.text(file='./test.txt.xz' suptws)
               compare.record(file='~.test.fba' suppad)
               report='~.test.out' offset
```

The example reads a UNIX text from an LZMA compressed file and compares the text records with records from an FBA data set. In order to make this compare work, trailing whitespace must be removed from the UNIX file and padding characters from the fixed-block host file. The ASA control characters are detached by default. Both texts are converted to UTF-8 and only the net record data is compared.

To simplify usage of this compare feature, there is a sample utility program SCFCUDIF. This program takes options and a list of filenames from command line for a logical compare of the resulting net data after decoding, decryption, decompression, character conversion and formatting as neutral FLAM elements. It uses the FLUCDIFF subprogram and writes the report to STDOUT. The first file is compared to all remaining files in the list and the program stops at the first error.

```
Parameters:
   :> SCFCUDIF "[-f] [-o] [-m] [-r] [reportfile] filename1 filename2 [filename3 ...  ↩
       filenameN]"
Options:
   -f Make a full compare of data or elements
   -o Print byte offsets and values to report file
   -m Minimize protocol output on STDERR
   -r The first given filename is used for the report file
```

You can use this utility program to compare every type of supported file format. It supports comma separated lists, so that it can be used on mainframe system. The example below is taken from the JOBLIB(SCFCUDIF) member of your FLAM installation.

```

```

The sample program SCFCUDIF is available on all platforms and the source is part of each FLAM installation (sample or SRCLIBC). You can simply adjust this program and use the provided build instructions (makefile or SBUILD job) to make your own utility.

ARGUMENTS

- `SWITCH:FULLY -Make a fully compare of data and elements [FALSE]`
- `SWITCH:OFFSET -Print byte offsets and values to report file [FALSE]`
- `STRING:REPORT='str'/STDOUT/STDERR -Generate a report file [NONE]`
  - `STDOUT -Write output to stdout`
  - `STDERR -Write output to stderr`

4.7.1 OVERLAY *READ*

SYNOPSIS

```
HELP:   Parameter to read data
PATH:   DIFF
TYPE:   OVERLAY
SYNTAX: READ.{BINARY()/CHARACTER()/TEXT()/XML()/RECORD()/FLAM4()/AUTO()/FILE='str'/STREAM/ ↩
    DUMMY}
```

DESCRIPTION With the overlay READ you can choose the kind of data (binary, text, character, record, FLAM4FILE, table, ...) to read or you can use *auto* for auto detection of the file/data format. For each read method, a dedicated behavior is implemented and the corresponding parameters can be used to detail the procedure. For example: If we know it is text and

the data is compressed then the data will be automatically de-compressed, because compressed data cannot be converted to text records. If the compressed data is, for example, Base64 encoded, then a decoding is done before decompression and so on. Based on the knowledge about the kind of data you want to read, we have simplified the way to automatically do the right things for you. Reading files transparently for each kind of data format can be very powerful.

After the read operation, you can choose the same or another write operation to convert your data between different platforms, file and data formats, character sets, compression, encryption or encoding methods.

Additionally you can calculate or verify checksums on the binary data stream direct after the read operation.

Below, you can find the currently supported data sources explained in a dedicated chapter.

4.7.2 OBJECT *BINARY*

SYNOPSIS

```
HELP:   Read binary data from a file
PATH:   DIFF.READ
TYPE:   OBJECT
SYNTAX: BINARY(FILE='str'/STREAM/DUMMY,BLKSIZE=num,DECODE[=NONE/FLAMFILE/FIODEC/DECRYP/ ↩
    CRYDEC/DECOMP/CMPDEC/ALWAYS],DECRYPT[{}...],FDECODE/COD.{},RLDECO,PRNCONTROL=DETACH/ ↩
    RETAIN/ERASE,SUBSYSTEM(),FRCBLK,REMOVE,PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK ↩
    )
```

DESCRIPTION Read binary operates on blocks of binary data. By default, no decompression or any other conversions are performed. These must explicitly be turned on using the respective parameters.

If the *decode* parameter is set, the data is automatically decrypted and uncompressed. If the encrypted or compressed data is encoded (e.g. Base64), it is decoded before uncompression/decryption. If a 4 byte record length format detected and decoding should be done, then the records are parsed before the next decoding step. If the uncompressed/decrypted data is encoded, it is only decoded if *decode* is set to *CMPDEC* or *ALWAYS*. By setting *decode* to one of the available values, you have further control over the enabled automatisms.

Binary data blocks are the simplest form of data type handled by FLAM. Binary data can only be accessed by byte offset. The structure of the data is unknown to FLAM.

Using only binary blocks with FLAM mainly equals the behavior and abilities of other encryption and compression tools. You may want to try one of the other available read methods to read structured data.

ARGUMENTS

- STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- NUMBER:DECODE=NONE/FLAMFILE/FIODEC/DECRYP/CRYDEC/DECOMP/CMPDEC/ALWAYS -Decode encoded, encrypted or compressed data [0=NONE]
  - NONE -No automated decoding
  - FLAMFILE -Transparent read of FLAMFILEs
  - FIODEC -Decoding after file IO before decryption
  - DECRYP -Decryption of encrypted data
  - CRYDEC -Decoding after decryption before decompression
  - DECOMP -Decompression of compressed data
  - CMPDEC -Decoding after decompression in front of formatting
  - ALWAYS -Always decode encoded data (causes the same as DECOMP)
- SWITCH:RLDECO -Activate an additional record length decoding [FALSE]
- SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.7.3 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size to read [AUTO]
PATH:   DIFF.READ.BINARY
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.7.4 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   DIFF.READ.BINARY
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.7.5 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   DIFF.READ.BINARY.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
 * In hexadecimal format (recommended): password=x'hex'
 * In ASCII format: password=a'ascii'
 * In EBCDIC format: password=e'ebcdic'
 * In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']`
  - `DEFAULT -FLAM4 default password`

4.7.6 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   DIFF.READ.BINARY.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- STRING:LIBRARY='str' -Library name for FKME ['libfkme']
- STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']
- STRING:PARAMETER='str' -Parameter for the call of FKME [”]

4.7.7 OVERLAY *FDECODE*

SYNOPSIS

```
HELP:   Additional enforced decoding [NONE]
PATH:   DIFF.READ.BINARY
TYPE:   OVERLAY
SYNTAX: FDECODE/COD.{BASE64/OPGP/BASE32/BASE16}
```

DESCRIPTION The overlay *fdecode* allows you to add an additional decoding step after auto decoding (*decode* parameter defined) was done.

On *read.bin()* with the *decode* parameter set, encrypted data (PGP, ...) is automatically decrypted and compressed data (GZIP, ...) is decompressed if the method is supported by FLAM. If the encrypted or compressed data is base encoded, it will decoded before decompression or decryption. However, there is no automatic base decoding of the decompressed or decrypted data since we are in binary mode and, therefore, it is impossible to guess whether you want to read base encoded or decoded data. The data might not even be base encoded, but just look like it is.

fdecode allows you to enforce base decoding after auto decoding by specifying the base to use. If you specify a base, but the data is not encoded with this base, you will get an error.

If you define *decode=CMPDEC* or *decode=ALWAYS* then a base decoding after decompression is done, if a base decoding is possible. This option can be used on behalf of *fdecode*, but auto detection of the correct base may fail. Therefore, it is suitable to use the *fdecode* parameter if you already know the base.

The enforcement of base decoding can even be used if no auto decoding is activated (i.e. *decode* parameter not specified).

See below for the decoding possibilities.

ARGUMENTS

- SWITCH:BASE64/OPGP -Base64 decoding (RFC4648)
- SWITCH:BASE32 -Base32 decoding (RFC4648)
- SWITCH:BASE16 -Base16 decoding (RFC4648)

4.7.8 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   DIFF.READ.BINARY
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.7.9 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   DIFF.READ.BINARY
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.7.10 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.READ.BINARY
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.7.11 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   DIFF.READ.BINARY
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- `NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]`
  - `MD5 -Message-digest algorithm 5 (128 bit)`
  - `RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)`
  - `RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)`

  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.12 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.READ.BINARY.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                            ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'                ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'               ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                  ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'       ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.13 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   DIFF.READ.BINARY
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.7.14 OBJECT *CHARACTER*

SYNOPSIS

```
HELP:   Read character data from a file
PATH:   DIFF.READ
TYPE:   OBJECT
SYNTAX: CHARACTER(FILE='str'/STREAM/DUMMY,BLKSIZE=num,CCSID='str'/DEFAULT/ASCII/EBCDIC/ ↩
    BOMUTF/BOMUCS,BOM,ENL2LF,DECODE=NONE/FIODEC/CRYDEC/CMPDEC/ALWAYS,DECRYPT[{}...], ↩
    PRNCONTROL=DETACH/RETAIN/ERASE,SUBSYSTEM(),FRCBLK,REMOVE,LANG='str',PLATFORM=WIN/UNX/ZOS ↩
    /USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read char operates on blocks of text data. If the input data is encoded (e.g. Base64), encrypted or compressed, it is automatically decoded, decrypted and/or decompressed. Through the *decode* parameter, you can limit the number of encoding layers to remove (for example to retrieve a Base64-encoded text in a GZIP file instead of the decoded text). If a 4 byte record length format is detected, the record lengths are removed to build a valid character block.

For decryption, you must provide the required decryption parameters, e.g. a key reference or passphrase. Several decryption methods can be enabled at once with the corresponding parameters and will be used to transparently read the data.

The character data is converted from the provided CCSID to UTF-8. If no CCSID is provided or auto detection fails, the default CCSID is determined in a system-dependent way in the following order:

- Stored CCSID from the file system (if available)
- From the environment variable LANG
- ISO-8859-1 (LATIN1) on ASCII and IBM-1148 (intl) on EBCDIC systems

The character data blocks can be converted to any code page on write. If you don't need character conversion, the binary version of reading files might be more suitable.

In contrast to the read text method, the read char method does not scan for line delimiters to build valid records. This means that the data will remain a chunk of character data.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`
- `NUMBER:DECODE=NONE/FIODEC/CRYDEC/CMPDEC/ALWAYS -Decode encoded data first (remove ba se encodings)`

  - NONE -No automated decoding
  - FIODEC -Decoding after file IO before decryption
  - CRYDEC -Decoding after decryption before decompression
  - CMPDEC -Decoding after decompression in front of formatting
  - ALWAYS -Always decode encoded data (causes the same as DECOMP)
- SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.7.15 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size for read operations [AUTO]
PATH:   DIFF.READ.CHARACTER
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.7.16 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID [auto detection]
PATH:   DIFF.READ.CHARACTER
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)

- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (auto-detect)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`
- `BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)`
- `BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)`

4.7.17 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   DIFF.READ.CHARACTER
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.7.18 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   DIFF.READ.CHARACTER
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.7.19 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   DIFF.READ.CHARACTER
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.7.20 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   DIFF.READ.CHARACTER.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
 * In hexadecimal format (recommended): password=x'hex'
 * In ASCII format: password=a'ascii'
 * In EBCDIC format: password=e'ebcdic'
 * In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']`
  - `DEFAULT -FLAM4 default password`

4.7.21 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   DIFF.READ.CHARACTER.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- STRING:LIBRARY='str' -Library name for FKME ['libfkme']
- STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']
- STRING:PARAMETER='str' -Parameter for the call of FKME [”]

4.7.22 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   DIFF.READ.CHARACTER
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- DETACH -Manage print control character as record attribute
- RETAIN -Leave print control character as part of the data record
- ERASE -Erase the print control character from the record

4.7.23 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   DIFF.READ.CHARACTER
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
SUBSYS(NAME='name' parm='keyword,keyword=value,value')
SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.7.24 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   DIFF.READ.CHARACTER
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.7.25 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.READ.CHARACTER
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- WIN -Choose Windows as platform (CRLF)
- UNX -Choose Unix derivative as platform (LF)
- ZOS -Choose ZOS as platform (NL)
- USS -Choose USS of ZOS as platform (NL)
- VSE -Choose VSE as platform (NL)
- BS2 -Choose BS2000-OSD as platform (NL)
- MAC -Choose MAC as platform (CR)

4.7.26 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   DIFF.READ.CHARACTER
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)

  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)

- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.27 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.READ.CHARACTER.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                            ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'                ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'               ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                  ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'       ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.28 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   DIFF.READ.CHARACTER
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.7.29 OBJECT *TEXT*

SYNOPSIS

```
HELP:   Read text data from a file
PATH:   DIFF.READ
TYPE:   OBJECT
SYNTAX: TEXT(FILE='str'/STREAM/DUMMY,BLKSIZE=num,RECLENGTH=num,SUPTWS,NELDLM,CCSID='str'/ ↩
    DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,BOM,ENL2LF,RPLFFD=num,PADCHAR=num,DECODE=NONE/FIODEC/ ↩
    CRYDEC/CMPDEC/ALWAYS,DECRYPT[{}...],PRNCONTROL=DETACH/RETAIN/ERASE,SUBSYSTEM(),FRCBLK, ↩
    REMOVE,BINERROR,CHRERROR,LANG='str',PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read text works on blocks of text data or records. First, the text data is converted to the UTF-8 character set. Then the text data is split into record and rest elements based on text delimiters or record length. The data must contain a valid text delimiter within the provided record/ line length. If the data contains 4 byte length fields, the length fields are used to build a block as list of records. If no delimiter is found, the record length are used to form the text records. If the data is block-oriented (no record length) and neither delimiters nor 4 byte length fields are found, the data will be wrapped into records of the provided record length.

A text record contains all text between two delimiters, but without trailing whitespace if SUPTWS used. Trailing whitespace, the delimiter and possibly some padding characters make up the rest element. Taken together, the record and the rest elements represent the original data.

List of valid text delimiters (UNICODE-Points):

- 0x00 (padding character (can be changed))
- 0x0A (carriage return (EBCDIC 0x25))
- 0x0D (line feed (EBCDIC(0x0D)))
- 0x0A, 0x0D (carriage return line feed)
- 0x0A, . . . , 0x0A, 0x0D (dirty delimiters)
- 0x0A, 0x0D, . . . , 0x0D (dirty delimiters)
- 0x0C (form feed (EBCDIC 0x22))
- 0x85 (new/next line (EBCDIC 0x15, optional if single byte ASCII))

0x85 are used for UTF-8 (C285) and EBCDIC (0x15) but in single byte ASCII code pages the NELDLM flag must be defined to scan for 0x85, because 0x85 normally used as currency sign accept for ISO code pages. To prevent EBCDIC(0x15) to UNICODE(0x85) at character conversion FLAM supports the ENL2LF and ELF2NL switches.

If the text data is detected as being compressed, it is decompressed automatically. For encodings like Base64, automated decoding is useful in most cases. However, if you want to read the base encoded data as text, you must set the *decode* parameter to the number of automatic decodings to perform. For a transparent decryption you, must provide the required parameters. Decryption requires at least a key reference as parameter. Several decryption methods can be enabled at once with the corresponding parameters. 4 byte length fields in the data are detected and cannot be part of valid text data. Therefore, these length fields are used automatically to build a list of records.

The character data is converted from the provided CCSID to UTF-8. If no CCSID is provided, auto detection is applied. If this not successful, a system dependent default CCSID is used: If available, the CCSID stored in the file system is used. If this is unset or not supported, the environment variable LANG is used. If this is also not set to a valid value, ISO-8859-1 (Latin-1) is used on ASCII and IBM-1148 (international) on EBCDIC systems.

While text formatting, form feed characters may optionally be replaced by empty records.

A padding character may be defined which is regarded as a special delimiter. Consecutive occurrences of this padding character are regarded as one delimiter, i.e. the text "some text_more text" will become two records "some text" and "more text". Not being a part of the original data, padding characters are simply dropped and are not part of the rest element.

With read.text() you can transparent read normal text files with delimiter in clear, compressed (GZIP, BZIP2, XZ(LZMA)), encrypted (FLAM, PGP) or encoded (Base64/32/16) form, normal record oriented data sets (FB(A/M)/VB(A/M)/VSAM/...) or FLAMFILEs.

ARGUMENTS

- STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- NUMBER:RECLENGTH=num -Maximum record length (Host) /line length (Unix/Win) for text parsing [512]
- SWITCH:SUPTWS -Suppress trailing whitespaces [FALSE]
- SWITCH:NELDLM -Activate NEL (0x85) as delimiter for ASCII character sets [FALSE]
- NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assuming n lines per page [0 =no replacement]
- NUMBER:PADCHAR=num -Padding character to be regarded as additional delimiter [0x00]
- NUMBER:DECODE=NONE/FIODEC/CRYDEC/CMPDEC/ALWAYS -Decode encoded data first (remove base encodings)
  - NONE -No automated decoding
  - FIODEC -Decoding after file IO before decryption
  - CRYDEC -Decoding after decryption before decompression
  - CMPDEC -Decoding after decompression in front of formatting
  - ALWAYS -Always decode encoded data (causes the same as DECOMP)
- SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]
- SWITCH:BINERROR -Enforce an error if binary data detected [FALSE]
- SWITCH:CHRERROR -Enforce an error if character data without delimiter detected [FALSE]

4.7.30 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size for read operations [AUTO]
PATH:   DIFF.READ.TEXT
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.7.31 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID [auto detection]
PATH:   DIFF.READ.TEXT
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- DEFAULT -Use default CCSID (auto-detect)
- ASCII -Use default ASCII CCSID (environment)
- EBCDIC -Use default EBCDIC CCSID (environment)
- BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)
- BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)

4.7.32 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   DIFF.READ.TEXT
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.7.33 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   DIFF.READ.TEXT
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.7.34 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   DIFF.READ.TEXT
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
 decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
 decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.7.35 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   DIFF.READ.TEXT.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']`
  - `DEFAULT -FLAM4 default password`

4.7.36 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   DIFF.READ.TEXT.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME [”]`

4.7.37 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   DIFF.READ.TEXT
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.7.38 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   DIFF.READ.TEXT
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.7.39 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   DIFF.READ.TEXT
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.7.40 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.READ.TEXT
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.7.41 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   DIFF.READ.TEXT
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.42 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.READ.TEXT.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                            ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'                ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'               ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                  ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'       ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.43 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   DIFF.READ.TEXT
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.7.44 OBJECT *XML*

SYNOPSIS

```
HELP:   Read XML data from a file
PATH:   DIFF.READ
TYPE:   OBJECT
SYNTAX: XML(FILE='str'/STREAM/DUMMY,BLKSIZE=num,NOCMNT,CCSID='str'/DEFAULT/ASCII/EBCDIC/ ↩
    BOMUTF/BOMUCS,BOM,ENL2LF,RPLFFD=num,DECRYPT[{}...],SUBSYSTEM(),FRCBLK,REMOVE,LANG='str', ↩
    PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read XML works on blocks of binary or text data. Character conversion takes place before processing the XML data. If no CCSID is provided, then auto detection is used. If UTF-8 is detected, character conversion is skipped. Line delimiters must be one of 0x0A, 0x0D or 0x0D0A (after conversion to UTF-8). If a CCSID is supplied, the character data is converted from the provided CCSID to UTF-8 before performing XML processing, using the supplied CCSID as source encoding. If the input data is encoded (e.g. Base64), encrypted, compressed or contains 4 byte length fields, it is automatically decoded, decrypted, and/or decompressed to build a valid XML data block. During parsing, all line delimiters are normalized to line feed characters (0xA) as defined by the XML specification.

The XML data is parsed using the Expat library and transformed into FLAM elements, which allows various formatting options when writing, including minimizing, pretty printing XML data and restoring an equivalent of the original document.

On some EBCDIC machines, if character conversion from an EBCDIC charset is used, the new line character (0x15) is not properly converted, causing the XML parser to fail. In this case, turn on the enl2lf to enable proper conversion of new lines (0x15) to line feeds (0xA).

If reading XML, the semantics of write modes (write.*()) change as follows:

- BINARY: The FLAM elements produced by this read mode are used to restore the original XML file as close as possible (standard formatting). The XML will be in UTF-8 and lines are delimited by line feed (0xA).
- CHARACTER: The FLAM elements are used to produce a minimized version of the XML data. Line breaks and whitespace are removed unless they are part of actual character data.
- TEXT: Human readable XML data with proper indentation is produced (pretty printing). Linebreaks are converted to a system-specific representation.
- XML: Produces XML data using the specified formatting method. The XML data will be in UTF-8 and lines are delimited by line feed (0xA) unless explicit character conversion is enabled.
- RECORD: Human readable XML data with proper indentation is produced (pretty printing). The produced XML data is separated into records, each record containing one line of XML.
- FLAM4: Same as RECORD, but stored in a FLAM4 file.

Known limitations:

- If the attributes inside an XML tag are separated by more than one space character, the additional space characters are dropped (as per XML specification).
- Due to a parser limitation, declared entities that appear inside attributes will be output in its substituted form.

ARGUMENTS

- STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- SWITCH:NOCMNT -Ignore XML comments [FALSE]
- NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assumin g n lines per page [0 =no replacement]
- SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.7.45 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size for read operations [AUTO]
PATH:   DIFF.READ.XML
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.7.46 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID [no character conversion, UTF-8/US-ASCII expected]
PATH:   DIFF.READ.XML
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- DEFAULT -Use default CCSID (auto-detect)
- ASCII -Use default ASCII CCSID (environment)
- EBCDIC -Use default EBCDIC CCSID (environment)
- BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)
- BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)

4.7.47 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   DIFF.READ.XML
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.7.48 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   DIFF.READ.XML
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.7.49 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   DIFF.READ.XML
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
 decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
 decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.7.50 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   DIFF.READ.XML.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['******']`
  - `DEFAULT -FLAM4 default password`

4.7.51 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   DIFF.READ.XML.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME [”]`

4.7.52 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   DIFF.READ.XML
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.7.53 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   DIFF.READ.XML
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.7.54 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.READ.XML
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.7.55 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check  of one way hash values of input data
PATH:   DIFF.READ.XML
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.56 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.READ.XML.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                            ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'                ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'               ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                  ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'       ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.57 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   DIFF.READ.XML
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.7.58 OBJECT *RECORD*

SYNOPSIS

```
HELP:   Read records from a file
PATH:   DIFF.READ
TYPE:   OBJECT
SYNTAX: RECORD(FILE='str'/STREAM/DUMMY,RECMODE=STOP/CUT/WRAP,PRNCONTROL=DETACH/RETAIN/ERASE ↩
    ,SUPPADDING,ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZE=num,RECFORMAT=NONE/BIN/TXT/ ↩
    DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX- ↩
    REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR ↩
    /FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/ ↩
    VRA/VRM,RECLENGTH=num,RECDELIM='bin'/CR-ASCII/LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF- ↩
    EBCDIC/NL-EBCDIC/CRLF-EBCDIC/CR-UTF08/LF-UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE ↩
    /NL-UTF16BE/CRLF-UTF16BE/CR-UTF16LE/LF-UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF- ↩
    UTF32BE/NL-UTF32BE/CRLF-UTF32BE/CR-UTF32LE/LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE,RECCOUNT= ↩
    num,LENFORMAT.{},KEYPOSITION=num,KEYLENGTH=num,DECODE[=NONE/FLAMFILE/FIODEC/ALWAYS], ↩
    DECRYPT[{}...],FDECODE/COD.{},CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,BOM, ↩
    SUBSYSTEM(),REMOVE,LANG='str',PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read record provides a list of records from a host-like data set format. If a record contains text data, you can mark it by defining the correct CCSID. This will result in character set conversion from the provided CCSID to UTF-8 for each record. If the CCSID is not provided, the record is handled as binary data.

The following options are supported:

- Stop, cut or wrap if a record is longer then provided record length
- Print control character handling as attributes
- Padding of fix record formats at write operations
- Suppress trailing padding characters for fix record formats at read
- Suppress trailing padding characters for variable record formats at write
- All kind of record formats and length including entry, index, keyed and relative data sets, including management of slot numbers
- Support of various file allocation, space parameters and subsystems

Decryption and decompression are only supported for FLAMFILEs at record I/O because other formats like GZIP or PGP are block-oriented. For example *read.text()* can be used to convert a compressed GZIP file into records.

If the write operation is text-oriented (write.char/text/xml()) or the CCSID is defined and the file for the read operation is a FLAMFILE then the records are read via the FLAM record interface. In all other cases, the decompression of a FLAMFILE can be enforced with the keyword *DECODE*. To enforce base decoding for each record, you must define one of the corresponding switches.

For not cataloged data sets or record-oriented file formats on non record-oriented platforms (WIN, UNIX, ...), you can define file organization, block size, record format and record length. If you work with cataloged files or FLAMFILEs, these parameters are not required.

For non-record-oriented platforms, you can define a lot of different length formats. If you don't specify a length format at read and a 4 byte record length format is detected (a FILEDATA=RECORD file is allocated with FILEDATA=BINARY), the length fields in the data are used to build the records.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`
- `NUMBER:RECMODE=STOP/CUT/WRAP -Mode used to read records [STOP]`
  - `STOP -Stop if record too long`
  - `CUT -Cut if record too long`
  - `WRAP -Wrap if record too long`
- `SWITCH:SUPPADDING -Suppress trailing padding characters (useful for fix records)`
- `NUMBER:ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [catalog/SEQ]`
  - `SEQ -Sequential data set [DEFAULT]`
  - `LIB -Library (ZOS-PDSE)`
  - `PDS -Partitioned data set (ZOS-PDS)`
  - `USS -UNIX files (path based)`
  - `EDS -Entry data set (VSAM-ESDS)`
  - `KDS -Keyed data set (VSAM-KSDS)`
  - `RDS -Relative data set (VSAM-RRDS)`
- `NUMBER:RECFORMAT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format for read operation [catalog/VAR]`

- NONE -No format defined
- BIN -Open -Binary (undefined) data (only the last record are shorter then record le ngth)
- TXT -Open -Text data with standard delimiter (result is a stripped variable length r ecords)
- DLM -Open -Records separated by a special delimiter with the result of variable len gth records
- VAR -Open -Variable length record (if necessary define length format)
- VAR-ASA -Open -Variable length record with ASA print control character (if necessar y define length format)
- VAR-MCC -Open -Variable length record with machine print control codes (if necessar y define length format)
- FIX -Open -Fix length record (file size =N * record length)
- FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
- FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
- VAR-REL -Open -Variable relative record (if necessary define length format)
- VAR-REL-ASA -Open -Variable relative record with ASA print control character (if ne cessary define length format)
- VAR-REL-MCC -Open -Variable relative record with machine print control codes (if ne cessary define length format)
- FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
- FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size = N * (record length + 9))
- FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size = N * (record length + 9))
- F -Host -Fixed-length, unblocked
- FA -Host -Fixed-length, unblocked, ASA print control characters
- FM -Host -Fixed-length, unblocked, machine print control codes
- FS -Host -Fixed-length, standard
- FSA -Host -Fixed-length, standard, ASA print control characters
- FSM -Host -Fixed-length, standard, machine print control codes
- FB -Host -Fixed-length, blocked
- FBA -Host -Fixed-length, blocked, ASA print control characters
- FBM -Host -Fixed-length, blocked, machine print control codes
- FBS -Host -Fixed-length, blocked, standard
- FBSA -Host -Fixed-length, blocked, standard, ASA print control characters
- FBSM -Host -Fixed-length, blocked, standard, machine print control codes
- FE -Host -Fixed-length, ESDS
- FEA -Host -Fixed-length, ESDS, ASA print control characters
- FEM -Host -Fixed-length, ESDS, machine print control codes
- FK -Host -Fixed-length, KSDS
- FKA -Host -Fixed-length, KSDS, ASA print control characters
- FKM -Host -Fixed-length, KSDS, machine print control codes
- FR -Host -Fixed-length, RRDS

  - FRA -Host -Fixed-length, RRDS, ASA print control characters
  - FRM -Host -Fixed-length, RRDS, machine print control codes
  - U -Host -Undefined-length
  - UA -Host -Undefined-length, ASA print control characters
  - UM -Host -Undefined-length, machine print control codes
  - V -Host -Variable
  - VA -Host -Variable, ASA print control characters
  - VM -Host -Variable, machine print control codes
  - VS -Host -Variable, spanned
  - VSA -Host -Variable, spanned, ASA print control characters
  - VSM -Host -Variable, spanned, machine print control codes
  - VB -Host -Variable, blocked
  - VBA -Host -Variable, blocked, ASA print control characters
  - VBM -Host -Variable, blocked, machine print control codes
  - VBS -Host -Variable, blocked, spanned
  - VBSA -Host -Variable, blocked, spanned, ASA print control characters
  - VBSM -Host -Variable, blocked, spanned, machine print control codes
  - VE -Host -Variable, ESDS
  - VEA -Host -Variable, ESDS, ASA print control characters
  - VEM -Host -Variable, ESDS, machine print control codes
  - VK -Host -Variable, KSDS
  - VKA -Host -Variable, KSDS, ASA print control characters
  - VKM -Host -Variable, KSDS, machine print control codes
  - VR -Host -Variable, RRDS
  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes
- NUMBER:RECLENGTH=num -Maximal record length for read operation [catalog/512]
- STRING:RECDELIM='bin'/CR-ASCII/LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF-EBCDIC/NL-EBCDIC/CRLF-EBCDIC/CR-UTF08/LF-UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE/NL-UTF16BE/CRLF-UTF16BE/CR-UTF16LE/LF-UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF-UTF32BE/NL-UTF32BE/CRLF-UTF32BE/CR-UTF32LE/LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE -Delimiter used to define end of record (if RECF=DLM used)
  - CR-ASCII -Delimiter:carriage return in ASCII (0x0D)
  - LF-ASCII -Delimiter:line feed in ASCII (0x0A)
  - NL-ASCII -Delimiter:new line in ASCII (0x85)
  - CRLF-ASCII -Delimiter:carriage return line feed in ASCII (0x0D0A)
  - CR-EBCDIC -Delimiter:carriage return in EBCDIC (0x0D)
  - LF-EBCDIC -Delimiter:line feed in EBCDIC (0x25)
  - NL-EBCDIC -Delimiter:new line in EBCDIC (0x15)
  - CRLF-EBCDIC -Delimiter:carriage return line feed in EBCDIC (0x0D25)
  - CR-UTF08 -Delimiter:carriage return in UTF-8 (0x0D)
  - LF-UTF08 -Delimiter:line feed in UTF-8 (0x0A)
  - NL-UTF08 -Delimiter:new line in UTF-8 (0xC285)
  - CRLF-UTF08 -Delimiter:carriage return line feed in UTF-8 (0x0D0A)

  - CR-UTF16BE -Delimiter:carriage return in UTF-16BE (0x000D)
  - LF-UTF16BE -Delimiter:line feed in UTF-16BE (0x000A)
  - NL-UTF16BE -Delimiter:new line in UTF-16BE (0x0085)
  - CRLF-UTF16BE -Delimiter:carriage return line feed in UTF-16BE (0x000D000A)
  - CR-UTF16LE -Delimiter:carriage return in UTF-16LE (0x0D00)
  - LF-UTF16LE -Delimiter:line feed in UTF-16LE (0x0A00)
  - NL-UTF16LE -Delimiter:new line in UTF-16LE (0x8500)
  - CRLF-UTF16LE -Delimiter:carriage return line feed in UTF-16LE (0x0D000A00)
  - CR-UTF32BE -Delimiter:carriage return in UTF-32BE (0x0000000D)
  - LF-UTF32BE -Delimiter:line feed in UTF-32BE (0x0000000A)
  - NL-UTF32BE -Delimiter:new line in UTF-32BE (0x00000085)
  - CRLF-UTF32BE -Delimiter:carriage return line feed in UTF-32BE (0x0000000D0000000A)
  - CR-UTF32LE -Delimiter:carriage return in UTF-32LE (0x0D000000)
  - LF-UTF32LE -Delimiter:line feed in UTF-32LE (0x0A000000)
  - NL-UTF32LE -Delimiter:new line in UTF-32LE (0x85000000)
  - CRLF-UTF32LE -Delimiter:carriage return line feed in UTF-32LE (0x0D0000000A000000)
- NUMBER:RECCOUNT=num -Amount of records handled in one read operation [catalog/128]
- NUMBER:KEYPOSITION=num -Key position (>=1) for index/key sequential data sets [0=NONE]
- NUMBER:KEYLENGTH=num -Key length (>=1) for index/key sequential data sets [if KEYPOS >0 then 8]
- NUMBER:DECODE=NONE/FLAMFILE/FIODEC/ALWAYS -Decode encoded, encrypted or compressed data [0=NONE]
  - NONE -No automated decoding
  - FLAMFILE -Transparent read of FLAMFILEs
  - FIODEC -Decoding after file IO (or read of FLAMFILE)
  - ALWAYS -Always decode encoded data (causes the same as FIODEC)
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.7.59 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   DIFF.READ.RECORD
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.7.60 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size for read operations [catalog/AUTO]
PATH:   DIFF.READ.RECORD
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.7.61 OVERLAY *LENFORMAT*

SYNOPSIS

```
HELP:   Format of the length field for open platforms [word]
PATH:   DIFF.READ.RECORD
TYPE:   OVERLAY
SYNTAX: LENFORMAT.{INTEGER()/STRING()/BCD()/HOST()}
```

DESCRIPTION The overlay can be used to define the format of the length filed on open platforms which don't support record formats by the data management system. For example: If you read records on a mainframe and you want to write the records as records (not as binary or text data) on WINDOWS or UNIX then the data will be prefixed with the record length. The format of this length field can be changed over this union.

This length specification will be only used for variable length records. The default mapping method between record formats will be map each host record format the corresponding open variable format.

If the length format not defined the length field will be written as 32 bit integer in platform dependent byte order. This behavior corresponds to the FILEDATA=RECORD parameter for USS files on z/OS. Means if you write a record oriented file to USS without any format changed over this overlay then you can read the record with another MVS application using FILEDATA=RECORD.

If you choose record format text or binary then this will correspond to FILEDATA=TEXT or FILEDATA=BINARY and the length format specification will be ignored. To enable record format mapping based on the data type you can set the environment variable below:

```
FL_RECORD_FORMAT_MAPPING=DATATYPE
```

If this environment variable set, then the length format is only used if you explicitly set a variable record format of the open systems.

Since version 5.1.8, FLAM can detect the 4 byte length fields (integer or host) inside a data stream. We recommend to use one of these formats to have more comfort when reading.

4.7.62 OBJECT *INTEGER*

SYNOPSIS

```
HELP:   Use integer format for the length field
PATH:   DIFF.READ.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: INTEGER(ENDIAN=SYSTEM/BIG/LITTLE,WIDTH=U16/U32/U64,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in integer format. You can define the byte order, the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [SYSTEM]
  - SYSTEM -Byte order of the current system
  - BIG -Big endian byte order
  - LITTLE -Little endian byte order
- NUMBER:WIDTH=U16/U32/U64 -Bit width/rate [B32]
  - U16 -16 bit unsigned integer
  - U32 -32 bit unsigned integer
  - U64 -64 bit unsigned integer
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.7.63 OBJECT *STRING*

SYNOPSIS

```
HELP:   Use string format for the length field
PATH:   DIFF.READ.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: STRING(CHRSET=SYSTEM/ASCII/EBCDIC,COUNT=C04/C05/C06,BASE=B10,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in string format (a character per digit). You can define the character set, the character count, the base and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:CHRSET=SYSTEM/ASCII/EBCDIC -Character set [SYSTEM]
  - SYSTEM -Use system code page
  - ASCII -Use ASCII (UTF-8) code page
  - EBCDIC -Use EBCDIC code page
- NUMBER:COUNT=C04/C05/C06 -Character count [C06]
  - C04 -4 character/digits
  - C05 -5 character/digits
  - C06 -6 character/digits
- NUMBER:BASE=B10 -String base [B10]
  - B10 -Base 10 (decimal)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.7.64 OBJECT *BCD*

SYNOPSIS

```
HELP:   Use BCD format for the length field
PATH:   DIFF.READ.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: BCD(WIDTH=U16/U24/U32,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in BCD format (4 Bit are one decimal digit). You can define the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:WIDTH=U16/U24/U32 -Bit width/rate [B32]
  - U16 -16 bit unsigned BCD (POV with 4 digits)
  - U24 -24 bit unsigned BCD (POV with 6 digits)
  - U32 -32 bit unsigned BCD (POV with 8 digits)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.7.65 OBJECT *HOST*

SYNOPSIS

```
HELP:   Use host format (LLxx) for the length field
PATH:   DIFF.READ.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: HOST(ENDIAN=SYSTEM/BIG/LITTLE,EXCLUDE)
```

DESCRIPTION The object defines the 4 byte host specific format (LLxx) of the length field. This format uses the first 16 bits as integer in big endian or little endian byte order to represent the record length including or excluding the record length field itself and the last 16 bits are set to zero when writing and ignored when reading.

For example, a record with 136 bytes content has a default length field (big endian, inclusive) of:

```
x'008D0000'
```

The default conforms to the length fields used by the data management system of MVS.

ARGUMENTS

- NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [BIG ENDIAN] for first 16 Bit
  - SYSTEM -Byte order of the current system
  - BIG -Big endian byte order
  - LITTLE -Little endian byte order
- SWITCH:EXCLUDE -Record length field not part of the length value [OFF]

4.7.66 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   DIFF.READ.RECORD
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.7.67 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   DIFF.READ.RECORD.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']
  - DEFAULT -FLAM4 default password

4.7.68 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   DIFF.READ.RECORD.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- STRING:LIBRARY='str' -Library name for FKME ['libfkme']
- STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']
- STRING:PARAMETER='str' -Parameter for the call of FKME [”]

4.7.69 OVERLAY *FDECODE*

SYNOPSIS

```
HELP:   Additional enforced decoding [NONE]
PATH:   DIFF.READ.RECORD
TYPE:   OVERLAY
SYNTAX: FDECODE/COD.{BASE64/OPGP/BASE32/BASE16}
```

DESCRIPTION The overlay *fdecode* allows you to enforce an additional decoding of the data after the auto decoding (*decode* parameter defined) was done. This is useful if the resulting data is still encoded. This may happen if a record was decoded from a FLAMFILE (read.record()) and the record is, for example, base encoded.

The forced decoding works on records and gives you the possibility to read files, which (for example) were encoded with:

```
'write.record(encode.base64())'
```

The enforcement of base decoding can even be used if no auto decoding is activated.

See below for the decoding possibilities.

ARGUMENTS

- `SWITCH:BASE64/OPGP -Base64 decoding (RFC4648)`
- `SWITCH:BASE32 -Base32 decoding (RFC4648)`
- `SWITCH:BASE16 -Base16 decoding (RFC4648)`

4.7.70 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID (enables text handling if set)
PATH:   DIFF.READ.RECORD
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
   flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (auto-detect)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`
- `BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)`
- `BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)`

4.7.71 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   DIFF.READ.RECORD
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.7.72 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   DIFF.READ.RECORD
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
SUBSYS(NAME='name' parm='keyword,keyword=value,value')
SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.7.73 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   DIFF.READ.RECORD
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.7.74 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.READ.RECORD
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- WIN -Choose Windows as platform (CRLF)
- UNX -Choose Unix derivative as platform (LF)
- ZOS -Choose ZOS as platform (NL)
- USS -Choose USS of ZOS as platform (NL)
- VSE -Choose VSE as platform (NL)
- BS2 -Choose BS2000-OSD as platform (NL)
- MAC -Choose MAC as platform (CR)

4.7.75 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   DIFF.READ.RECORD
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)

  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.76 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.READ.RECORD.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                            ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'                ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'               ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                  ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'       ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.77 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   DIFF.READ.RECORD
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.7.78 OBJECT *FLAM4*

SYNOPSIS

```
HELP:   Read records from a FLAM4FILE
PATH:   DIFF.READ
TYPE:   OBJECT
SYNTAX: FLAM4(FILE='str',MEMBER='str',PRNCONTROL=DETACH/RETAIN/ERASE,RECCOUNT=num,RECLENGTH ↩
    =num,TRUNCATE,PASSWORD/CRYPTOKEY='bin'/DEFAULT,KMLIBRARY='str',KMFUNCTION='str', ↩
    KMPARAMETER='str',CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,BOM,REMOVE,LANG='str', ↩
    PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH())
```

DESCRIPTION Read FLAM4 works with a list of records from a FLAMFILE of version 4 or older. If such a record is text data, you can mark this by defining the correct CCSID. Such a definition will result in character set conversion from the provided CCSID to UTF-8 for each record. If no CCSID is provided, the record will be handled as binary data.

For FLAMFILE archives, these features are supported:

- Read all members or a dedicated member of the archive
- Print control character handling defined by a dedicated mode
- Pass phrase based or FKME based decryption technology
- Free definition of the amount of records handled as one chunk

User I/O, pre- and post-processing exits are not supported. If these features are required, please use the extended variant (XCMP/XCNV) of this command.

If you need, for example, Base64 decoding for records of a FLAMFILE, please use read.record(fdecode.base64). Read.record() can transparently read FLAMFILEs, supports base decoding and other useful features for record handling.

ARGUMENTS

- `STRING:FILE='str' -Name/URL of the FLAMFILE to read [required]`
- `STRING:MEMBER='str' -Name or Index of the member in the FLAMFILE to read`
- `NUMBER:RECCOUNT=num -Amount of records handled in one read operation [128]`
- `NUMBER:RECLENGTH=num -Maximal record length for read operation [512]`
- `SWITCH:TRUNCATE -Allow truncation of records [FALSE]`

- STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to decrypt the FLAMFILE ["]
  - DEFAULT -FLAM4 default password
- STRING:KMLIBRARY='str' -Library name for FKME ["]
- STRING:KMFUNCTION='str' -Function name of FKME ["]
- STRING:KMPARAMETER='str' -Parameter for the call of FKME ["]
- SWITCH:REMOVE -Remove FLAMFILE after successful processing [FALSE]

4.7.79 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   DIFF.READ.FLAM4
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- DETACH -Manage print control character as record attribute
- RETAIN -Leave print control character as part of the data record
- ERASE -Erase the print control character from the record

4.7.80 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID (enables text handling if set)
PATH:   DIFF.READ.FLAM4
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)

- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (auto-detect)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`
- `BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)`
- `BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)`

4.7.81 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   DIFF.READ.FLAM4
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.7.82 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   DIFF.READ.FLAM4
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.7.83 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.READ.FLAM4
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- WIN -Choose Windows as platform (CRLF)
- UNX -Choose Unix derivative as platform (LF)
- ZOS -Choose ZOS as platform (NL)
- USS -Choose USS of ZOS as platform (NL)
- VSE -Choose VSE as platform (NL)
- BS2 -Choose BS2000-OSD as platform (NL)
- MAC -Choose MAC as platform (CR)

4.7.84 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   DIFF.READ.FLAM4
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)

  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.85 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.READ.FLAM4.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                              ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'                  ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'                 ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                    ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'         ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.86 OBJECT *AUTO*

SYNOPSIS

```
HELP:   Read binary, text or XML data record- or block-oriented
PATH:   DIFF.READ
TYPE:   OBJECT
SYNTAX: AUTO(FILE='str'/STREAM/DUMMY,BLKSIZE=num,RECLENGTH=num,RPLFFD=num,CCSID='str'/ ↩
    DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,BOM,ENL2LF,DECRYPT[{}...],REMOVE,LANG='str',PLATFORM= ↩
    WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read *auto* attempts to identify the file format automatically and reads the file with the most appropriate parameters. Detection works on a best effort basis and should not be relied on.

For example, "read.auto() write.record()" can convert every supported file format to records on mainframe systems. This is utilized in the FLVIEW and FLVEDIT ISPF command.

With read.auto(), you can transparently read normal XML or text files with delimiters, encoded (BASE16/32/64), encrypted (FLAMFILE) or compressed (GZIP, BZIP2, XZ(LZMA)) files (even nested!), normal record-oriented data sets (FB(A/M)/VB(A/M)/VSA FLAMFILEs or binary data. If the data contains 4 byte length fields after I/O, decryption, decompression then a list of records is built to form one block. This allows you for example to read a file transparently from USS which was created with FILEDATA=RECORD and now allocated with FILEDATA=BINARY or a record length based format from a ZIP archive. The read.auto() function was mainly implemented to display (FLVIEW) the content of a file. The detection may fail on some files, especially for very small files.

Generally: If you know the type of data (binary, char, text, xml, flam, charset/ccsid), it is recommended to use the corresponding read method and set the known parameters to prevent false detections. On the other side, if you know nothing about your data or you get data from different sources, read.auto() might be very helpful.

To simplify the use of the read.auto() function, the mechanisms below are implemented:

- The EBCDIC new line to ASCII line feed management is activated. This means that, for methods that support it, the ENL2LF flag is set for read methods and the ELF2NL flag is set for the write methods.
- Suppression of trailing whitespace is activated for text records.
- The print control character handling is set to erase print control characters from records.
- Subsystem specifications are not supported
- Base encodings are always decoded, if the decoding results in useful data. (i.e. text files in an a Base64 encoded Gzip file are decoded and uncompressed automatically). Optional CRC verification and verification of the trailer (if ASCII armor) is done if the CHECK switch is enabled.
- Record length format detection is enabled and used after I/O (a wrong static file allocation on z/OS was done (FILEDATA=BINARY instead of FILEDATA=RECORD) or a file with variable record length was read on Windows or UNIX platforms), after decryption (data containing record length fields were encrypted), after decompression (data with record length fields was compressed (ZIP)) and after base decoding (if data with length fields was encode).

To read ASCII text files with NEL delimiter or to use specific formatting options, please use read.text(). To keep and manage record attributes, please use read.record(). The read.auto() function is designed to be very powerful, using the procedures suited best and, at the same time, to keep it as simple as possible.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`
- `NUMBER:RECLENGTH=num -Maximum record length (Host) /line length (Unix/Win) for text p arsing [512]`

- NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assumin g n lines per page [0 =no replacement]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.7.87 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size to read [AUTO]
PATH:   DIFF.READ.AUTO
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.7.88 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID [auto detection]
PATH:   DIFF.READ.AUTO
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE

- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (auto-detect)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`
- `BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)`
- `BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)`

4.7.89 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   DIFF.READ.AUTO
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.7.90 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   DIFF.READ.AUTO
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.7.91 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   DIFF.READ.AUTO
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.7.92 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   DIFF.READ.AUTO.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
 * In hexadecimal format (recommended): password=x'hex'
 * In ASCII format: password=a'ascii'
 * In EBCDIC format: password=e'ebcdic'
 * In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']`
  - `DEFAULT -FLAM4 default password`

4.7.93 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   DIFF.READ.AUTO.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- STRING:LIBRARY='str' -Library name for FKME ['libfkme']
- STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']
- STRING:PARAMETER='str' -Parameter for the call of FKME [”]

4.7.94 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   DIFF.READ.AUTO
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.7.95 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.READ.AUTO
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- WIN -Choose Windows as platform (CRLF)
- UNX -Choose Unix derivative as platform (LF)
- ZOS -Choose ZOS as platform (NL)

- USS -Choose USS of ZOS as platform (NL)
- VSE -Choose VSE as platform (NL)
- BS2 -Choose BS2000-OSD as platform (NL)
- MAC -Choose MAC as platform (CR)

4.7.96 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   DIFF.READ.AUTO
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)

  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.97 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.READ.AUTO.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.98 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   DIFF.READ.AUTO
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.7.99 PARAMETER *FILE*

SYNOPSIS

```
HELP:   Abbreviation for read.auto(file=...)
PATH:   DIFF.READ
TYPE:   STRING
SYNTAX: FILE='str'/STREAM/DUMMY
```

DESCRIPTION `read.file='name.txt'` is an abbreviation for `read.auto(file='name.txt')` without any further parameters. Please have a look at read.auto() for more information.

`read.file` use all data format detection capabilities of FLAM to read transparent byte streams or records form a data source, which can be an encoded (e.g. Base64), encrypted (e.g. PGP) and/or compressed (GZIP/BZIP2/XZ) file, a FLAMFILE, other archive or a normal clear data set.

The data stream after I/O and after each conversion can contain several kinds of 4 byte length field formats. This is also detected and the data is interpreted as blocks containing various records for the next layer.

SELECTIONS

- `STREAM -Read from stdin or write to stdout`
- `DUMMY -Read EOF or write nothing`

4.7.100 OVERLAY *COMPARE*

SYNOPSIS

```
HELP:   Parameter to compare with data read
PATH:   DIFF
TYPE:   OVERLAY
SYNTAX: COMPARE.{BINARY()/CHARACTER()/TEXT()/XML()/RECORD()/FLAM4()/AUTO()/FILE='str'/ ↩
    STREAM/DUMMY}
```

DESCRIPTION With the overlay READ you can choose the kind of data (binary, text, character, record, FLAM4FILE, table, ...) to read or you can use *auto* for auto detection of the file/data format. For each read method, a dedicated behavior is implemented and the corresponding parameters can be used to detail the procedure. For example: If we know it is text and the data is compressed then the data will be automatically de-compressed, because compressed data cannot be converted to text records. If the compressed data is, for example, Base64 encoded, then a decoding is done before decompression and so on. Based on the knowledge about the kind of data you want to read, we have simplified the way to automatically do the right things for you. Reading files transparently for each kind of data format can be very powerful.

After the read operation, you can choose the same or another write operation to convert your data between different platforms, file and data formats, character sets, compression, encryption or encoding methods.

Additionally you can calculate or verify checksums on the binary data stream direct after the read operation.

Below, you can find the currently supported data sources explained in a dedicated chapter.

4.7.101 OBJECT *BINARY*

SYNOPSIS

```
HELP:   Read binary data from a file
PATH:   DIFF.COMPARE
TYPE:   OBJECT
SYNTAX: BINARY(FILE='str'/STREAM/DUMMY,BLKSIZE=num,DECODE[=NONE/FLAMFILE/FIODEC/DECRYP/ ↩
    CRYDEC/DECOMP/CMPDEC/ALWAYS],DECRYPT[{}...],FDECODE/COD.{},RLDECO,PRNCONTROL=DETACH/ ↩
    RETAIN/ERASE,SUBSYSTEM(),FRCBLK,REMOVE,PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK ↩
    )
```

DESCRIPTION Read binary operates on blocks of binary data. By default, no decompression or any other conversions are performed. These must explicitly be turned on using the respective parameters.

If the *decode* parameter is set, the data is automatically decrypted and uncompressed. If the encrypted or compressed data is encoded (e.g. Base64), it is decoded before uncompression/decryption. If a 4 byte record length format detected and decoding should be done, then the records are parsed before the next decoding step. If the uncompressed/decrypted data is encoded, it is only decoded if *decode* is set to *CMPDEC* or *ALWAYS*. By setting *decode* to one of the available values, you have further control over the enabled automatisms.

Binary data blocks are the simplest form of data type handled by FLAM. Binary data can only be accessed by byte offset. The structure of the data is unknown to FLAM.

Using only binary blocks with FLAM mainly equals the behavior and abilities of other encryption and compression tools. You may want to try one of the other available read methods to read structured data.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`
- `NUMBER:DECODE=NONE/FLAMFILE/FIODEC/DECRYP/CRYDEC/DECOMP/CMPDEC/ALWAYS -Decode encoded, encrypted or compressed data [0=NONE]`
  - `NONE -No automated decoding`
  - `FLAMFILE -Transparent read of FLAMFILEs`
  - `FIODEC -Decoding after file IO before decryption`
  - `DECRYP -Decryption of encrypted data`
  - `CRYDEC -Decoding after decryption before decompression`
  - `DECOMP -Decompression of compressed data`
  - `CMPDEC -Decoding after decompression in front of formatting`
  - `ALWAYS -Always decode encoded data (causes the same as DECOMP)`
- `SWITCH:RLDECO -Activate an additional record length decoding [FALSE]`
- `SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]`
- `SWITCH:REMOVE -Remove original file after successful processing [FALSE]`

4.7.102 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size to read [AUTO]
PATH:   DIFF.COMPARE.BINARY
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the

data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.7.103 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   DIFF.COMPARE.BINARY
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.7.104 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   DIFF.COMPARE.BINARY.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['******']`
  - `DEFAULT -FLAM4 default password`

4.7.105 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   DIFF.COMPARE.BINARY.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME [”]`

4.7.106 OVERLAY *FDECODE*

SYNOPSIS

```
HELP:   Additional enforced decoding [NONE]
PATH:   DIFF.COMPARE.BINARY
TYPE:   OVERLAY
SYNTAX: FDECODE/COD.{BASE64/OPGP/BASE32/BASE16}
```

DESCRIPTION The overlay *fdecode* allows you to add an additional decoding step after auto decoding (*decode* parameter defined) was done.

On *read.bin()* with the *decode* parameter set, encrypted data (PGP, ...) is automatically decrypted and compressed data (GZIP, ...) is decompressed if the method is supported by FLAM. If the encrypted or compressed data is base encoded, it will decoded before decompression or decryption. However, there is no automatic base decoding of the decompressed or decrypted data since we are in binary mode and, therefore, it is impossible to guess whether you want to read base encoded or decoded data. The data might not even be base encoded, but just look like it is.

fdecode allows you to enforce base decoding after auto decoding by specifying the base to use. If you specify a base, but the data is not encoded with this base, you will get an error.

If you define *decode=CMPDEC* or *decode=ALWAYS* then a base decoding after decompression is done, if a base decoding is possible. This option can be used on behalf of *fdecode*, but auto detection of the correct base may fail. Therefore, it is suitable to use the *fdecode* parameter if you already know the base.

The enforcement of base decoding can even be used if no auto decoding is activated (i.e. *decode* parameter not specified).

See below for the decoding possibilities.

ARGUMENTS

- `SWITCH:BASE64/OPGP -Base64 decoding (RFC4648)`
- `SWITCH:BASE32 -Base32 decoding (RFC4648)`
- `SWITCH:BASE16 -Base16 decoding (RFC4648)`

4.7.107 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   DIFF.COMPARE.BINARY
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.7.108 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   DIFF.COMPARE.BINARY
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.7.109 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.COMPARE.BINARY
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`

- ZOS -Choose ZOS as platform (NL)
- USS -Choose USS of ZOS as platform (NL)
- VSE -Choose VSE as platform (NL)
- BS2 -Choose BS2000-OSD as platform (NL)
- MAC -Choose MAC as platform (CR)

4.7.110 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   DIFF.COMPARE.BINARY
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant

  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.111 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.COMPARE.BINARY.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.112 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   DIFF.COMPARE.BINARY
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.7.113 OBJECT *CHARACTER*

SYNOPSIS

```
HELP:   Read character data from a file
PATH:   DIFF.COMPARE
TYPE:   OBJECT
SYNTAX: CHARACTER(FILE='str'/STREAM/DUMMY,BLKSIZE=num,CCSID='str'/DEFAULT/ASCII/EBCDIC/ ↩
    BOMUTF/BOMUCS,BOM,ENL2LF,DECODE=NONE/FIODEC/CRYDEC/CMPDEC/ALWAYS,DECRYPT[{}...], ↩
    PRNCONTROL=DETACH/RETAIN/ERASE,SUBSYSTEM(),FRCBLK,REMOVE,LANG='str',PLATFORM=WIN/UNX/ZOS ↩
    /USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read char operates on blocks of text data. If the input data is encoded (e.g. Base64), encrypted or compressed, it is automatically decoded, decrypted and/or decompressed. Through the *decode* parameter, you can limit the number of encoding layers to remove (for example to retrieve a Base64-encoded text in a GZIP file instead of the decoded text). If a 4 byte record length format is detected, the record lengths are removed to build a valid character block.

For decryption, you must provide the required decryption parameters, e.g. a key reference or passphrase. Several decryption methods can be enabled at once with the corresponding parameters and will be used to transparently read the data.

The character data is converted from the provided CCSID to UTF-8. If no CCSID is provided or auto detection fails, the default CCSID is determined in a system-dependent way in the following order:

- Stored CCSID from the file system (if available)
- From the environment variable LANG
- ISO-8859-1 (LATIN1) on ASCII and IBM-1148 (intl) on EBCDIC systems

The character data blocks can be converted to any code page on write. If you don't need character conversion, the binary version of reading files might be more suitable.

In contrast to the read text method, the read char method does not scan for line delimiters to build valid records. This means that the data will remain a chunk of character data.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`
- `NUMBER:DECODE=NONE/FIODEC/CRYDEC/CMPDEC/ALWAYS -Decode encoded data first (remove ba se encodings)`
  - `NONE -No automated decoding`
  - `FIODEC -Decoding after file IO before decryption`
  - `CRYDEC -Decoding after decryption before decompression`
  - `CMPDEC -Decoding after decompression in front of formatting`
  - `ALWAYS -Always decode encoded data (causes the same as DECOMP)`
- `SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]`
- `SWITCH:REMOVE -Remove original file after successful processing [FALSE]`

4.7.114 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size for read operations [AUTO]
PATH:   DIFF.COMPARE.CHARACTER
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.7.115 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID [auto detection]
PATH:   DIFF.COMPARE.CHARACTER
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- DEFAULT -Use default CCSID (auto-detect)
- ASCII -Use default ASCII CCSID (environment)
- EBCDIC -Use default EBCDIC CCSID (environment)
- BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)
- BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)

4.7.116 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   DIFF.COMPARE.CHARACTER
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.7.117 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   DIFF.COMPARE.CHARACTER
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.7.118 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   DIFF.COMPARE.CHARACTER
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.7.119 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   DIFF.COMPARE.CHARACTER.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']`
  - `DEFAULT -FLAM4 default password`

4.7.120 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   DIFF.COMPARE.CHARACTER.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME [”]`

4.7.121 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   DIFF.COMPARE.CHARACTER
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.7.122 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   DIFF.COMPARE.CHARACTER
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.7.123 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   DIFF.COMPARE.CHARACTER
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.7.124 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.COMPARE.CHARACTER
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.7.125 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   DIFF.COMPARE.CHARACTER
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.126 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.COMPARE.CHARACTER.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.127 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   DIFF.COMPARE.CHARACTER
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.7.128 OBJECT *TEXT*

SYNOPSIS

```
HELP:   Read text data from a file
PATH:   DIFF.COMPARE
TYPE:   OBJECT
SYNTAX: TEXT(FILE='str'/STREAM/DUMMY,BLKSIZE=num,RECLENGTH=num,SUPTWS,NELDLM,CCSID='str'/ ↩
    DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,BOM,ENL2LF,RPLFFD=num,PADCHAR=num,DECODE=NONE/FIODEC/ ↩
    CRYDEC/CMPDEC/ALWAYS,DECRYPT[{}...],PRNCONTROL=DETACH/RETAIN/ERASE,SUBSYSTEM(),FRCBLK, ↩
    REMOVE,BINERROR,CHRERROR,LANG='str',PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read text works on blocks of text data or records. First, the text data is converted to the UTF-8 character set. Then the text data is split into record and rest elements based on text delimiters or record length. The data must contain a valid text delimiter within the provided record/ line length. If the data contains 4 byte length fields, the length fields are used to build a block as list of records. If no delimiter is found, the record length are used to form the text records. If the data is block-oriented (no record length) and neither delimiters nor 4 byte length fields are found, the data will be wrapped into records of the provided record length.

A text record contains all text between two delimiters, but without trailing whitespace if SUPTWS used. Trailing whitespace, the delimiter and possibly some padding characters make up the rest element. Taken together, the record and the rest elements represent the original data.

List of valid text delimiters (UNICODE-Points):

- 0x00 (padding character (can be changed))
- 0x0A (carriage return (EBCDIC 0x25))
- 0x0D (line feed (EBCDIC(0x0D)))
- 0x0A, 0x0D (carriage return line feed)
- 0x0A, . . . , 0x0A, 0x0D (dirty delimiters)
- 0x0A, 0x0D, . . . , 0x0D (dirty delimiters)
- 0x0C (form feed (EBCDIC 0x22))
- 0x85 (new/next line (EBCDIC 0x15, optional if single byte ASCII))

0x85 are used for UTF-8 (C285) and EBCDIC (0x15) but in single byte ASCII code pages the NELDLM flag must be defined to scan for 0x85, because 0x85 normally used as currency sign accept for ISO code pages. To prevent EBCDIC(0x15) to UNICODE(0x85) at character conversion FLAM supports the ENL2LF and ELF2NL switches.

If the text data is detected as being compressed, it is decompressed automatically. For encodings like Base64, automated decoding is useful in most cases. However, if you want to read the base encoded data as text, you must set the *decode* parameter to the number of automatic decodings to perform. For a transparent decryption you, must provide the required parameters. Decryption requires at least a key reference as parameter. Several decryption methods can be enabled at once with the corresponding parameters. 4 byte length fields in the data are detected and cannot be part of valid text data. Therefore, these length fields are used automatically to build a list of records.

The character data is converted from the provided CCSID to UTF-8. If no CCSID is provided, auto detection is applied. If this not successful, a system dependent default CCSID is used: If available, the CCSID stored in the file system is used. If this is unset or not supported, the environment variable LANG is used. If this is also not set to a valid value, ISO-8859-1 (Latin-1) is used on ASCII and IBM-1148 (international) on EBCDIC systems.

While text formatting, form feed characters may optionally be replaced by empty records.

A padding character may be defined which is regarded as a special delimiter. Consecutive occurrences of this padding character are regarded as one delimiter, i.e. the text "some text_more text" will become two records "some text" and "more text". Not being a part of the original data, padding characters are simply dropped and are not part of the rest element.

With read.text() you can transparent read normal text files with delimiter in clear, compressed (GZIP, BZIP2, XZ(LZMA)), encrypted (FLAM, PGP) or encoded (Base64/32/16) form, normal record oriented data sets (FB(A/M)/VB(A/M)/VSAM/...) or FLAMFILEs.

ARGUMENTS

- STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- NUMBER:RECLENGTH=num -Maximum record length (Host) /line length (Unix/Win) for text parsing [512]
- SWITCH:SUPTWS -Suppress trailing whitespaces [FALSE]
- SWITCH:NELDLM -Activate NEL (0x85) as delimiter for ASCII character sets [FALSE]
- NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assuming n lines per page [0 =no replacement]
- NUMBER:PADCHAR=num -Padding character to be regarded as additional delimiter [0x00]
- NUMBER:DECODE=NONE/FIODEC/CRYDEC/CMPDEC/ALWAYS -Decode encoded data first (remove base encodings)
  - NONE -No automated decoding
  - FIODEC -Decoding after file IO before decryption
  - CRYDEC -Decoding after decryption before decompression
  - CMPDEC -Decoding after decompression in front of formatting
  - ALWAYS -Always decode encoded data (causes the same as DECOMP)
- SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]
- SWITCH:BINERROR -Enforce an error if binary data detected [FALSE]
- SWITCH:CHRERROR -Enforce an error if character data without delimiter detected [FALSE]

4.7.129 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size for read operations [AUTO]
PATH:   DIFF.COMPARE.TEXT
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection

services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.7.130 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID [auto detection]
PATH:   DIFF.COMPARE.TEXT
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
   flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (auto-detect)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`
- `BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)`
- `BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)`

4.7.131 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   DIFF.COMPARE.TEXT
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.7.132 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   DIFF.COMPARE.TEXT
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.7.133 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   DIFF.COMPARE.TEXT
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.7.134 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   DIFF.COMPARE.TEXT.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']
  - DEFAULT -FLAM4 default password

4.7.135 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   DIFF.COMPARE.TEXT.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME [”]`

4.7.136 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   DIFF.COMPARE.TEXT
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- DETACH -Manage print control character as record attribute
- RETAIN -Leave print control character as part of the data record
- ERASE -Erase the print control character from the record

4.7.137 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   DIFF.COMPARE.TEXT
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- STRING:NAME='str' -Name of the subsystem
- STRING:PARM='str' -Parameter list for the subsystem (comma separated)

4.7.138 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   DIFF.COMPARE.TEXT
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.7.139 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.COMPARE.TEXT
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.7.140 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   DIFF.COMPARE.TEXT
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.141 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.COMPARE.TEXT.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                            ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'                ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'               ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                  ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'       ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.142 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   DIFF.COMPARE.TEXT
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.7.143 OBJECT *XML*

SYNOPSIS

```
HELP:   Read XML data from a file
PATH:   DIFF.COMPARE
TYPE:   OBJECT
SYNTAX: XML(FILE='str'/STREAM/DUMMY,BLKSIZE=num,NOCMNT,CCSID='str'/DEFAULT/ASCII/EBCDIC/ ↩
    BOMUTF/BOMUCS,BOM,ENL2LF,RPLFFD=num,DECRYPT[{}...],SUBSYSTEM(),FRCBLK,REMOVE,LANG='str', ↩
    PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read XML works on blocks of binary or text data. Character conversion takes place before processing the XML data. If no CCSID is provided, then auto detection is used. If UTF-8 is detected, character conversion is skipped. Line delimiters must be one of 0x0A, 0x0D or 0x0D0A (after conversion to UTF-8). If a CCSID is supplied, the character data is converted from the provided CCSID to UTF-8 before performing XML processing, using the supplied CCSID as source encoding. If the input data is encoded (e.g. Base64), encrypted, compressed or contains 4 byte length fields, it is automatically decoded, decrypted, and/or decompressed to build a valid XML data block. During parsing, all line delimiters are normalized to line feed characters (0xA) as defined by the XML specification.

The XML data is parsed using the Expat library and transformed into FLAM elements, which allows various formatting options when writing, including minimizing, pretty printing XML data and restoring an equivalent of the original document.

On some EBCDIC machines, if character conversion from an EBCDIC charset is used, the new line character (0x15) is not properly converted, causing the XML parser to fail. In this case, turn on the enl2lf to enable proper conversion of new lines (0x15) to line feeds (0xA).

If reading XML, the semantics of write modes (write.*()) change as follows:

- BINARY: The FLAM elements produced by this read mode are used to restore the original XML file as close as possible (standard formatting). The XML will be in UTF-8 and lines are delimited by line feed (0xA).
- CHARACTER: The FLAM elements are used to produce a minimized version of the XML data. Line breaks and whitespace are removed unless they are part of actual character data.
- TEXT: Human readable XML data with proper indentation is produced (pretty printing). Linebreaks are converted to a system-specific representation.
- XML: Produces XML data using the specified formatting method. The XML data will be in UTF-8 and lines are delimited by line feed (0xA) unless explicit character conversion is enabled.
- RECORD: Human readable XML data with proper indentation is produced (pretty printing). The produced XML data is separated into records, each record containing one line of XML.
- FLAM4: Same as RECORD, but stored in a FLAM4 file.

Known limitations:

- If the attributes inside an XML tag are separated by more than one space character, the additional space characters are dropped (as per XML specification).
- Due to a parser limitation, declared entities that appear inside attributes will be output in its substituted form.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`
- `SWITCH:NOCMNT -Ignore XML comments [FALSE]`
- `NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assumin g n lines per page [0 =no replacement]`
- `SWITCH:FRCBLK -Enforce block orientation on record oriented devices [FALSE]`
- `SWITCH:REMOVE -Remove original file after successful processing [FALSE]`

4.7.144 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size for read operations [AUTO]
PATH:   DIFF.COMPARE.XML
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection

services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.7.145 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID [no character conversion, UTF-8/US-ASCII expected]
PATH:   DIFF.COMPARE.XML
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 $\Rightarrow$ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE $\Rightarrow$ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
   flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (auto-detect)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`
- `BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)`
- `BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)`

4.7.146 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   DIFF.COMPARE.XML
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.7.147 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   DIFF.COMPARE.XML
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.7.148 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   DIFF.COMPARE.XML
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.7.149 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   DIFF.COMPARE.XML.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']
  - DEFAULT -FLAM4 default password

4.7.150 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   DIFF.COMPARE.XML.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME [”]`

4.7.151 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   DIFF.COMPARE.XML
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
SUBSYS(NAME='name' parm='keyword,keyword=value,value')
SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- STRING:NAME='str' -Name of the subsystem
- STRING:PARM='str' -Parameter list for the subsystem (comma separated)

4.7.152 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   DIFF.COMPARE.XML
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.7.153 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.COMPARE.XML
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- WIN -Choose Windows as platform (CRLF)
- UNX -Choose Unix derivative as platform (LF)
- ZOS -Choose ZOS as platform (NL)
- USS -Choose USS of ZOS as platform (NL)

- VSE -Choose VSE as platform (NL)
- BS2 -Choose BS2000-OSD as platform (NL)
- MAC -Choose MAC as platform (CR)

4.7.154 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check  of one way hash values of input data
PATH:   DIFF.COMPARE.XML
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)

- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.155 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.COMPARE.XML.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.156 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   DIFF.COMPARE.XML
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.7.157 OBJECT *RECORD*

SYNOPSIS

```
HELP:   Read records from a file
PATH:   DIFF.COMPARE
TYPE:   OBJECT
SYNTAX: RECORD(FILE='str'/STREAM/DUMMY,RECMODE=STOP/CUT/WRAP,PRNCONTROL=DETACH/RETAIN/ERASE ↩
    ,SUPPADDING,ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS,BLKSIZE=num,RECFORMAT=NONE/BIN/TXT/ ↩
    DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX- ↩
    REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR ↩
    /FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/ ↩
    VRA/VRM,RECLENGTH=num,RECDELIM='bin'/CR-ASCII/LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF- ↩
    EBCDIC/NL-EBCDIC/CRLF-EBCDIC/CR-UTF08/LF-UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE ↩
    /NL-UTF16BE/CRLF-UTF16BE/CR-UTF16LE/LF-UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF- ↩
    UTF32BE/NL-UTF32BE/CRLF-UTF32BE/CR-UTF32LE/LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE,RECCOUNT= ↩
    num,LENFORMAT.{},KEYPOSITION=num,KEYLENGTH=num,DECODE[=NONE/FLAMFILE/FIODEC/ALWAYS], ↩
    DECRYPT[{}...],FDECODE/COD.{},CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,BOM, ↩
    SUBSYSTEM(),REMOVE,LANG='str',PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read record provides a list of records from a host-like data set format. If a record contains text data, you can mark it by defining the correct CCSID. This will result in character set conversion from the provided CCSID to UTF-8 for each record. If the CCSID is not provided, the record is handled as binary data.

The following options are supported:

- Stop, cut or wrap if a record is longer then provided record length
- Print control character handling as attributes
- Padding of fix record formats at write operations
- Suppress trailing padding characters for fix record formats at read
- Suppress trailing padding characters for variable record formats at write
- All kind of record formats and length including entry, index, keyed and relative data sets, including management of slot numbers
- Support of various file allocation, space parameters and subsystems

Decryption and decompression are only supported for FLAMFILEs at record I/O because other formats like GZIP or PGP are block-oriented. For example *read.text()* can be used to convert a compressed GZIP file into records.

If the write operation is text-oriented (write.char/text/xml()) or the CCSID is defined and the file for the read operation is a FLAMFILE then the records are read via the FLAM record interface. In all other cases, the decompression of a FLAMFILE can be enforced with the keyword *DECODE*. To enforce base decoding for each record, you must define one of the corresponding switches.

For not cataloged data sets or record-oriented file formats on non record-oriented platforms (WIN, UNIX, ...), you can define file organization, block size, record format and record length. If you work with cataloged files or FLAMFILEs, these parameters are not required.

For non-record-oriented platforms, you can define a lot of different length formats. If you don't specify a length format at read and a 4 byte record length format is detected (a FILEDATA=RECORD file is allocated with FILEDATA=BINARY), the length fields in the data are used to build the records.

ARGUMENTS

- `STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]`
  - `STREAM -Read from stdin or write to stdout`
  - `DUMMY -Read EOF or write nothing`

- NUMBER:RECMODE=STOP/CUT/WRAP -Mode used to read records [STOP]
  - STOP -Stop if record too long
  - CUT -Cut if record too long
  - WRAP -Wrap if record too long
- SWITCH:SUPPADDING -Suppress trailing padding characters (useful for fix records)
- NUMBER:ORGANIZATION=SEQ/LIB/PDS/USS/EDS/KDS/RDS -File organization [catalog/SEQ]
  - SEQ -Sequential data set [DEFAULT]
  - LIB -Library (ZOS-PDSE)
  - PDS -Partitioned data set (ZOS-PDS)
  - USS -UNIX files (path based)
  - EDS -Entry data set (VSAM-ESDS)
  - KDS -Keyed data set (VSAM-KSDS)
  - RDS -Relative data set (VSAM-RRDS)
- NUMBER:RECFORMAT=NONE/BIN/TXT/DLM/VAR/VAR-ASA/VAR-MCC/FIX/FIX-ASA/FIX-MCC/VAR-REL/VAR-REL-ASA/VAR-REL-MCC/FIX-REL/FIX-REL-ASA/FIX-REL-MCC/F/FA/FM/FS/FSA/FSM/FB/FBA/FBM/FBS/FBSA/FBSM/FE/FEA/FEM/FK/FKA/FKM/FR/FRA/FRM/U/UA/UM/V/VA/VM/VS/VSA/VSM/VB/VBA/VBM/VBS/VBSA/VBSM/VE/VEA/VEM/VK/VKA/VKM/VR/VRA/VRM -Record format for read operation [catalog/VAR]
  - NONE -No format defined
  - BIN -Open -Binary (undefined) data (only the last record are shorter then record length)
  - TXT -Open -Text data with standard delimiter (result is a stripped variable length records)
  - DLM -Open -Records separated by a special delimiter with the result of variable length records
  - VAR -Open -Variable length record (if necessary define length format)
  - VAR-ASA -Open -Variable length record with ASA print control character (if necessary define length format)
  - VAR-MCC -Open -Variable length record with machine print control codes (if necessary define length format)
  - FIX -Open -Fix length record (file size =N * record length)
  - FIX-ASA -Open -Fix length record with ASA print control character (file size =N * (record length + 1))
  - FIX-MCC -Open -Fix length record with machine print control codes (file size =N * (record length + 1))
  - VAR-REL -Open -Variable relative record (if necessary define length format)
  - VAR-REL-ASA -Open -Variable relative record with ASA print control character (if necessary define length format)
  - VAR-REL-MCC -Open -Variable relative record with machine print control codes (if necessary define length format)
  - FIX-REL -Open -Fix relative record (file size =N * (record length + 8))
  - FIX-REL-ASA -Open -Fix relative record with ASA print control character (file size =N * (record length + 9))
  - FIX-REL-MCC -Open -Fix relative record with machine print control codes (file size =N * (record length + 9))

- F -Host -Fixed-length, unblocked
- FA -Host -Fixed-length, unblocked, ASA print control characters
- FM -Host -Fixed-length, unblocked, machine print control codes
- FS -Host -Fixed-length, standard
- FSA -Host -Fixed-length, standard, ASA print control characters
- FSM -Host -Fixed-length, standard, machine print control codes
- FB -Host -Fixed-length, blocked
- FBA -Host -Fixed-length, blocked, ASA print control characters
- FBM -Host -Fixed-length, blocked, machine print control codes
- FBS -Host -Fixed-length, blocked, standard
- FBSA -Host -Fixed-length, blocked, standard, ASA print control characters
- FBSM -Host -Fixed-length, blocked, standard, machine print control codes
- FE -Host -Fixed-length, ESDS
- FEA -Host -Fixed-length, ESDS, ASA print control characters
- FEM -Host -Fixed-length, ESDS, machine print control codes
- FK -Host -Fixed-length, KSDS
- FKA -Host -Fixed-length, KSDS, ASA print control characters
- FKM -Host -Fixed-length, KSDS, machine print control codes
- FR -Host -Fixed-length, RRDS
- FRA -Host -Fixed-length, RRDS, ASA print control characters
- FRM -Host -Fixed-length, RRDS, machine print control codes
- U -Host -Undefined-length
- UA -Host -Undefined-length, ASA print control characters
- UM -Host -Undefined-length, machine print control codes
- V -Host -Variable
- VA -Host -Variable, ASA print control characters
- VM -Host -Variable, machine print control codes
- VS -Host -Variable, spanned
- VSA -Host -Variable, spanned, ASA print control characters
- VSM -Host -Variable, spanned, machine print control codes
- VB -Host -Variable, blocked
- VBA -Host -Variable, blocked, ASA print control characters
- VBM -Host -Variable, blocked, machine print control codes
- VBS -Host -Variable, blocked, spanned
- VBSA -Host -Variable, blocked, spanned, ASA print control characters
- VBSM -Host -Variable, blocked, spanned, machine print control codes
- VE -Host -Variable, ESDS
- VEA -Host -Variable, ESDS, ASA print control characters
- VEM -Host -Variable, ESDS, machine print control codes
- VK -Host -Variable, KSDS
- VKA -Host -Variable, KSDS, ASA print control characters
- VKM -Host -Variable, KSDS, machine print control codes
- VR -Host -Variable, RRDS

  - VRA -Host -Variable, RRDS, ASA print control characters
  - VRM -Host -Variable, RRDS, machine print control codes
- NUMBER:RECLENGTH=num -Maximal record length for read operation [catalog/512]
- STRING:RECDELIM='bin'/CR-ASCII/LF-ASCII/NL-ASCII/CRLF-ASCII/CR-EBCDIC/LF-EBCDIC/NL-EBCDIC/CRLF-EBCDIC/CR-UTF08/LF-UTF08/NL-UTF08/CRLF-UTF08/CR-UTF16BE/LF-UTF16BE/NL-UTF16BE/CRLF-UTF16BE/CR-UTF16LE/LF-UTF16LE/NL-UTF16LE/CRLF-UTF16LE/CR-UTF32BE/LF-UTF32BE/NL-UTF32BE/CRLF-UTF32BE/CR-UTF32LE/LF-UTF32LE/NL-UTF32LE/CRLF-UTF32LE -Delimiter used to define end of record (if RECF=DLM used)
  - CR-ASCII -Delimiter:carriage return in ASCII (0x0D)
  - LF-ASCII -Delimiter:line feed in ASCII (0x0A)
  - NL-ASCII -Delimiter:new line in ASCII (0x85)
  - CRLF-ASCII -Delimiter:carriage return line feed in ASCII (0x0D0A)
  - CR-EBCDIC -Delimiter:carriage return in EBCDIC (0x0D)
  - LF-EBCDIC -Delimiter:line feed in EBCDIC (0x25)
  - NL-EBCDIC -Delimiter:new line in EBCDIC (0x15)
  - CRLF-EBCDIC -Delimiter:carriage return line feed in EBCDIC (0x0D25)
  - CR-UTF08 -Delimiter:carriage return in UTF-8 (0x0D)
  - LF-UTF08 -Delimiter:line feed in UTF-8 (0x0A)
  - NL-UTF08 -Delimiter:new line in UTF-8 (0xC285)
  - CRLF-UTF08 -Delimiter:carriage return line feed in UTF-8 (0x0D0A)
  - CR-UTF16BE -Delimiter:carriage return in UTF-16BE (0x000D)
  - LF-UTF16BE -Delimiter:line feed in UTF-16BE (0x000A)
  - NL-UTF16BE -Delimiter:new line in UTF-16BE (0x0085)
  - CRLF-UTF16BE -Delimiter:carriage return line feed in UTF-16BE (0x000D000A)
  - CR-UTF16LE -Delimiter:carriage return in UTF-16LE (0x0D00)
  - LF-UTF16LE -Delimiter:line feed in UTF-16LE (0x0A00)
  - NL-UTF16LE -Delimiter:new line in UTF-16LE (0x8500)
  - CRLF-UTF16LE -Delimiter:carriage return line feed in UTF-16LE (0x0D000A00)
  - CR-UTF32BE -Delimiter:carriage return in UTF-32BE (0x0000000D)
  - LF-UTF32BE -Delimiter:line feed in UTF-32BE (0x0000000A)
  - NL-UTF32BE -Delimiter:new line in UTF-32BE (0x00000085)
  - CRLF-UTF32BE -Delimiter:carriage return line feed in UTF-32BE (0x0000000D0000000A)
  - CR-UTF32LE -Delimiter:carriage return in UTF-32LE (0x0D000000)
  - LF-UTF32LE -Delimiter:line feed in UTF-32LE (0x0A000000)
  - NL-UTF32LE -Delimiter:new line in UTF-32LE (0x85000000)
  - CRLF-UTF32LE -Delimiter:carriage return line feed in UTF-32LE (0x0D0000000A000000)
- NUMBER:RECCOUNT=num -Amount of records handled in one read operation [catalog/128]
- NUMBER:KEYPOSITION=num -Key position (>=1) for index/key sequential data sets [0=NONE]
- NUMBER:KEYLENGTH=num -Key length (>=1) for index/key sequential data sets [if KEYPOS>0 then 8]
- NUMBER:DECODE=NONE/FLAMFILE/FIODEC/ALWAYS -Decode encoded, encrypted or compressed data [0=NONE]
  - NONE -No automated decoding
  - FLAMFILE -Transparent read of FLAMFILEs
  - FIODEC -Decoding after file IO (or read of FLAMFILE)
  - ALWAYS -Always decode encoded data (causes the same as FIODEC)
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.7.158 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   DIFF.COMPARE.RECORD
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- `DETACH -Manage print control character as record attribute`
- `RETAIN -Leave print control character as part of the data record`
- `ERASE -Erase the print control character from the record`

4.7.159 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size for read operations [catalog/AUTO]
PATH:   DIFF.COMPARE.RECORD
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.7.160 OVERLAY *LENFORMAT*

SYNOPSIS

```
HELP:   Format of the length field for open platforms [word]
PATH:   DIFF.COMPARE.RECORD
TYPE:   OVERLAY
SYNTAX: LENFORMAT.{INTEGER()/STRING()/BCD()/HOST()}
```

DESCRIPTION The overlay can be used to define the format of the length filed on open platforms which don't support record formats by the data management system. For example: If you read records on a mainframe and you want to write the records as records (not as binary or text data) on WINDOWS or UNIX then the data will be prefixed with the record length. The format of this length field can be changed over this union.

This length specification will be only used for variable length records. The default mapping method between record formats will be map each host record format the corresponding open variable format.

If the length format not defined the length field will be written as 32 bit integer in platform dependent byte order. This behavior corresponds to the FILEDATA=RECORD parameter for USS files on z/OS. Means if you write a record oriented file to USS without any format changed over this overlay then you can read the record with another MVS application using FILEDATA=RECORD.

If you choose record format text or binary then this will correspond to FILEDATA=TEXT or FILEDATA=BINARY and the length format specification will be ignored. To enable record format mapping based on the data type you can set the environment variable below:

```
FL_RECORD_FORMAT_MAPPING=DATATYPE
```

If this environment variable set, then the length format is only used if you explicitly set a variable record format of the open systems.

Since version 5.1.8, FLAM can detect the 4 byte length fields (integer or host) inside a data stream. We recommend to use one of these formats to have more comfort when reading.

4.7.161 OBJECT *INTEGER*

SYNOPSIS

```
HELP:   Use integer format for the length field
PATH:   DIFF.COMPARE.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: INTEGER(ENDIAN=SYSTEM/BIG/LITTLE,WIDTH=U16/U32/U64,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in integer format. You can define the byte order, the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [SYSTEM]
  - SYSTEM -Byte order of the current system
  - BIG -Big endian byte order
  - LITTLE -Little endian byte order
- NUMBER:WIDTH=U16/U32/U64 -Bit width/rate [B32]
  - U16 -16 bit unsigned integer
  - U32 -32 bit unsigned integer
  - U64 -64 bit unsigned integer
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.7.162 OBJECT *STRING*

SYNOPSIS

```
HELP:   Use string format for the length field
PATH:   DIFF.COMPARE.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: STRING(CHRSET=SYSTEM/ASCII/EBCDIC,COUNT=C04/C05/C06,BASE=B10,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in string format (a character per digit). You can define the character set, the character count, the base and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:CHRSET=SYSTEM/ASCII/EBCDIC -Character set [SYSTEM]
  - SYSTEM -Use system code page
  - ASCII -Use ASCII (UTF-8) code page
  - EBCDIC -Use EBCDIC code page
- NUMBER:COUNT=C04/C05/C06 -Character count [C06]
  - C04 -4 character/digits
  - C05 -5 character/digits
  - C06 -6 character/digits
- NUMBER:BASE=B10 -String base [B10]
  - B10 -Base 10 (decimal)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.7.163 OBJECT *BCD*

SYNOPSIS

```
HELP:   Use BCD format for the length field
PATH:   DIFF.COMPARE.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: BCD(WIDTH=U16/U24/U32,INCLUDE)
```

DESCRIPTION The object defines the binary representation of the length filed in BCD format (4 Bit are one decimal digit). You can define the bit width and if the length field will be part of the length value or not.

ARGUMENTS

- NUMBER:WIDTH=U16/U24/U32 -Bit width/rate [B32]
  - U16 -16 bit unsigned BCD (POV with 4 digits)
  - U24 -24 bit unsigned BCD (POV with 6 digits)
  - U32 -32 bit unsigned BCD (POV with 8 digits)
- SWITCH:INCLUDE -Include length field in length value [OFF]

4.7.164 OBJECT *HOST*

SYNOPSIS

```
HELP:   Use host format (LLxx) for the length field
PATH:   DIFF.COMPARE.RECORD.LENFORMAT
TYPE:   OBJECT
SYNTAX: HOST(ENDIAN=SYSTEM/BIG/LITTLE,EXCLUDE)
```

DESCRIPTION The object defines the 4 byte host specific format (LLxx) of the length field. This format uses the first 16 bits as integer in big endian or little endian byte order to represent the record length including or excluding the record length field itself and the last 16 bits are set to zero when writing and ignored when reading.

For example, a record with 136 bytes content has a default length field (big endian, inclusive) of:

```
   x'008D0000'
```

The default conforms to the length fields used by the data management system of MVS.

ARGUMENTS

- NUMBER:ENDIAN=SYSTEM/BIG/LITTLE -Byte order [BIG ENDIAN] for first 16 Bit
  - SYSTEM -Byte order of the current system
  - BIG -Big endian byte order
  - LITTLE -Little endian byte order
- SWITCH:EXCLUDE -Record length field not part of the length value [OFF]

4.7.165 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   DIFF.COMPARE.RECORD
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
 decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
 decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.7.166 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   DIFF.COMPARE.RECORD.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
* In hexadecimal format (recommended): password=x'hex'
* In ASCII format: password=a'ascii'
* In EBCDIC format: password=e'ebcdic'
* In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']`

  – `DEFAULT -FLAM4 default password`

4.7.167 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   DIFF.COMPARE.RECORD.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:LIBRARY='str' -Library name for FKME ['libfkme']`
- `STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']`
- `STRING:PARAMETER='str' -Parameter for the call of FKME [”]`

4.7.168 OVERLAY *FDECODE*

SYNOPSIS

```
HELP:   Additional enforced decoding [NONE]
PATH:   DIFF.COMPARE.RECORD
TYPE:   OVERLAY
SYNTAX: FDECODE/COD.{BASE64/OPGP/BASE32/BASE16}
```

DESCRIPTION The overlay *fdecode* allows you to enforce an additional decoding of the data after the auto decoding (*decode* parameter defined) was done. This is useful if the resulting data is still encoded. This may happen if a record was decoded from a FLAMFILE (read.record()) and the record is, for example, base encoded.

The forced decoding works on records and gives you the possibility to read files, which (for example) were encoded with:

```
'write.record(encode.base64())'
```

The enforcement of base decoding can even be used if no auto decoding is activated.

See below for the decoding possibilities.

ARGUMENTS

- `SWITCH:BASE64/OPGP -Base64 decoding (RFC4648)`
- `SWITCH:BASE32 -Base32 decoding (RFC4648)`
- `SWITCH:BASE16 -Base16 decoding (RFC4648)`

4.7.169 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID (enables text handling if set)
PATH:   DIFF.COMPARE.RECORD
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (auto-detect)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`
- `BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)`
- `BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)`

4.7.170 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   DIFF.COMPARE.RECORD
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.7.171 OBJECT *SUBSYSTEM*

SYNOPSIS

```
HELP:   Specification for FIO subsystem (only for z/OS)
PATH:   DIFF.COMPARE.RECORD
TYPE:   OBJECT
SYNTAX: SUBSYSTEM(NAME='str',PARM='str')
```

DESCRIPTION The subsystem definition object can be used to activate a subsystem on platforms like z/OS® where a subsystem can be used for file I/O.

To activate a subsystem, the name and a parameter list can be specified.

The name is a string (up to 4 characters on z/OS). The parameter list must be comma separated. For example:

```
   SUBSYS(NAME='name' parm='keyword,keyword=value,value')
   SUBSYS(NAME='FLAM' parm='IG10,TRACE,MODE=ADC,MAXR=2048')
```

Please don't use spaces in the parameter list. The parameter list can be empty. This is the default if the *PARM* keyword is not defined. The default for the subsystem name is *FLAM*. The example below

```
   SUBSYS()
```

activates the FLAM subsystem without any parameter. The subsystem name and parameter are converted to uppercase. To write a file through the subsystem the file must already exist. The allocation of a new file is not possible.

ARGUMENTS

- `STRING:NAME='str' -Name of the subsystem`
- `STRING:PARM='str' -Parameter list for the subsystem (comma separated)`

4.7.172 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   DIFF.COMPARE.RECORD
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.7.173 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.COMPARE.RECORD
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.7.174 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   DIFF.COMPARE.RECORD
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.175 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.COMPARE.RECORD.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                            ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'                ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'               ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                  ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'       ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.176 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   DIFF.COMPARE.RECORD
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.7.177 OBJECT *FLAM4*

SYNOPSIS

```
HELP:   Read records from a FLAM4FILE
PATH:   DIFF.COMPARE
TYPE:   OBJECT
SYNTAX: FLAM4(FILE='str',MEMBER='str',PRNCONTROL=DETACH/RETAIN/ERASE,RECCOUNT=num,RECLENGTH ↩
    =num,TRUNCATE,PASSWORD/CRYPTOKEY='bin'/DEFAULT,KMLIBRARY='str',KMFUNCTION='str', ↩
    KMPARAMETER='str',CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,BOM,REMOVE,LANG='str', ↩
    PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH())
```

DESCRIPTION Read FLAM4 works with a list of records from a FLAMFILE of version 4 or older. If such a record is text data, you can mark this by defining the correct CCSID. Such a definition will result in character set conversion from the provided CCSID to UTF-8 for each record. If no CCSID is provided, the record will be handled as binary data.

For FLAMFILE archives, these features are supported:

- Read all members or a dedicated member of the archive
- Print control character handling defined by a dedicated mode
- Pass phrase based or FKME based decryption technology
- Free definition of the amount of records handled as one chunk

User I/O, pre- and post-processing exits are not supported. If these features are required, please use the extended variant (XCMP/XCNV) of this command.

If you need, for example, Base64 decoding for records of a FLAMFILE, please use read.record(fdecode.base64). Read.record() can transparently read FLAMFILEs, supports base decoding and other useful features for record handling.

ARGUMENTS

- STRING:FILE='str' -Name/URL of the FLAMFILE to read [required]
- STRING:MEMBER='str' -Name or Index of the member in the FLAMFILE to read
- NUMBER:RECCOUNT=num -Amount of records handled in one read operation [128]
- NUMBER:RECLENGTH=num -Maximal record length for read operation [512]
- SWITCH:TRUNCATE -Allow truncation of records [FALSE]
- STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to decrypt the FLAMFILE ["]
  - DEFAULT -FLAM4 default password
- STRING:KMLIBRARY='str' -Library name for FKME ["]
- STRING:KMFUNCTION='str' -Function name of FKME ["]
- STRING:KMPARAMETER='str' -Parameter for the call of FKME ["]
- SWITCH:REMOVE -Remove FLAMFILE after successful processing [FALSE]

4.7.178 PARAMETER *PRNCONTROL*

SYNOPSIS

```
HELP:   Print control character handling [DETACH]
PATH:   DIFF.COMPARE.FLAM4
TYPE:   NUMBER
SYNTAX: PRNCONTROL=DETACH/RETAIN/ERASE
```

DESCRIPTION Print control characters handling is a special feature for record- oriented files of mainframe systems. For print control, the ASA or machine control characters are often coded in the first byte of the record.

For read operations, you can choose whether this byte will be part of the data (RETAIN), handled as separate attribute (DETACH) or ignored (ERASE). With DETACH, you can later (on write) decide how to handle the control character. With RETAIN, the character will be part of the data and with ERASE the control character is lost.

If block I/O is used DETACH works like ERASE because the print control character cannot be stored as attribute for the records of a block. This is only possible with record I/O, where the print control character, slot numbers and other attributes are be manageable per record.

If you want to read a block of data often, such print control characters are not useful and to transfer such files to an non-mainframe system can result in some strange behavior if there are ASA or MCC control characters in the data. On the other side, sometimes you may want to read a file with ASA and must write records with MCC control characters. In such cases, it is useful to still know the value. In most cases, DETACH is the best choice and is the default for print control handling.

SELECTIONS

- DETACH -Manage print control character as record attribute
- RETAIN -Leave print control character as part of the data record
- ERASE -Erase the print control character from the record

4.7.179 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID (enables text handling if set)
PATH:   DIFF.COMPARE.FLAM4
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)
- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- DEFAULT –Use default CCSID (auto-detect)
- ASCII –Use default ASCII CCSID (environment)
- EBCDIC –Use default EBCDIC CCSID (environment)
- BOMUTF –Determine the correct UTF CCSID from byte order mark (BOM)
- BOMUCS –Determine the correct UCS CCSID from byte order mark (BOM)

4.7.180 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   DIFF.COMPARE.FLAM4
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.7.181 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   DIFF.COMPARE.FLAM4
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.7.182 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.COMPARE.FLAM4
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- `WIN -Choose Windows as platform (CRLF)`
- `UNX -Choose Unix derivative as platform (LF)`
- `ZOS -Choose ZOS as platform (NL)`
- `USS -Choose USS of ZOS as platform (NL)`
- `VSE -Choose VSE as platform (NL)`
- `BS2 -Choose BS2000-OSD as platform (NL)`
- `MAC -Choose MAC as platform (CR)`

4.7.183 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   DIFF.COMPARE.FLAM4
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- `NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]`

  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant
  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.184 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.COMPARE.FLAM4.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.185 OBJECT *AUTO*

SYNOPSIS

```
HELP:   Read binary, text or XML data record- or block-oriented
PATH:   DIFF.COMPARE
TYPE:   OBJECT
SYNTAX: AUTO(FILE='str'/STREAM/DUMMY,BLKSIZE=num,RECLENGTH=num,RPLFFD=num,CCSID='str'/ ↩
    DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS,BOM,ENL2LF,DECRYPT[{}...],REMOVE,LANG='str',PLATFORM= ↩
    WIN/UNX/ZOS/USS/VSE/BS2/MAC,HASH(),CHECK)
```

DESCRIPTION Read *auto* attempts to identify the file format automatically and reads the file with the most appropriate parameters. Detection works on a best effort basis and should not be relied on.

For example, "read.auto() write.record()" can convert every supported file format to records on mainframe systems. This is utilized in the FLVIEW and FLVEDIT ISPF command.

With read.auto(), you can transparently read normal XML or text files with delimiters, encoded (BASE16/32/64), encrypted (FLAMFILE) or compressed (GZIP, BZIP2, XZ(LZMA)) files (even nested!), normal record-oriented data sets (FB(A/M)/VB(A/M)/VSA FLAMFILEs or binary data. If the data contains 4 byte length fields after I/O, decryption, decompression then a list of records is built to form one block. This allows you for example to read a file transparently from USS which was created with FILEDATA=RECORD and now allocated with FILEDATA=BINARY or a record length based format from a ZIP archive. The read.auto() function was mainly implemented to display (FLVIEW) the content of a file. The detection may fail on some files, especially for very small files.

Generally: If you know the type of data (binary, char, text, xml, flam, charset/ccsid), it is recommended to use the corresponding read method and set the known parameters to prevent false detections. On the other side, if you know nothing about your data or you get data from different sources, read.auto() might be very helpful.

To simplify the use of the read.auto() function, the mechanisms below are implemented:

- The EBCDIC new line to ASCII line feed management is activated. This means that, for methods that support it, the ENL2LF flag is set for read methods and the ELF2NL flag is set for the write methods.
- Suppression of trailing whitespace is activated for text records.
- The print control character handling is set to erase print control characters from records.
- Subsystem specifications are not supported
- Base encodings are always decoded, if the decoding results in useful data. (i.e. text files in an a Base64 encoded Gzip file are decoded and uncompressed automatically). Optional CRC verification and verification of the trailer (if ASCII armor) is done if the CHECK switch is enabled.
- Record length format detection is enabled and used after I/O (a wrong static file allocation on z/OS was done (FILEDATA=BINARY instead of FILEDATA=RECORD) or a file with variable record length was read on Windows or UNIX platforms), after decryption (data containing record length fields were encrypted), after decompression (data with record length fields was compressed (ZIP)) and after base decoding (if data with length fields was encode).

To read ASCII text files with NEL delimiter or to use specific formatting options, please use read.text(). To keep and manage record attributes, please use read.record(). The read.auto() function is designed to be very powerful, using the procedures suited best and, at the same time, to keep it as simple as possible.

ARGUMENTS

- STRING:FILE='str'/STREAM/DUMMY -Name/URL of file to read ["==stdin]
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing
- NUMBER:RECLENGTH=num -Maximum record length (Host) /line length (Unix/Win) for text parsing [512]
- NUMBER:RPLFFD=num -Replace form feeds, filling rest of page with blank lines assuming n lines per page [0 =no replacement]
- SWITCH:REMOVE -Remove original file after successful processing [FALSE]

4.7.186 PARAMETER *BLKSIZE*

SYNOPSIS

```
HELP:   Block size to read [AUTO]
PATH:   DIFF.COMPARE.AUTO
TYPE:   NUMBER
SYNTAX: BLKSIZE=num
```

DESCRIPTION The block size parameter defines the amount of data which is processed at a time. Conversion and formatting steps can result in a expansion or contraction of the internal block size.

A block can contain a bunch of records before and elements after formatting. It depends on the block size how many records/elements fit into one data block.

If this parameter is not specified (0), the default block size is determined from the system / catalog. On block I/O, this default has a minimum of 64 KiB or 4 records to ensure that all format detection algorithms get sufficient amounts of data.

If a block size is specified, these automatism are disabled. If you set a smaller block size, the internal memory consumption would be smaller, the CPU time for auto detection is smaller, but more CPU time is spent for the overhead used to handle the data units. Auto detection is only performed on the first block of data. If you define a small block size and use auto detection services, it is more likely that detection fails or yields incorrect results. A large block size results in better format detection and usually in a faster operation.

By default or if you define a block size greater than 64 KiB on record-oriented systems (MVS), the internal block size is greater than the valid block size range and at file allocation the optimal block size is used, because in this case we use a block size of 0 for dynamic allocation. If the internal block size within a valid range, we use the next possible block size. The *FALLOC* object can be used to overrule the block size used for file allocation. The internal block size is only the default, if no block size for file allocation is defined.

Only if you use record I/O without any conversions which expand or shrink the data chunks, you can be sure that the block size used for reading is the same used for writing.

4.7.187 PARAMETER *CCSID*

SYNOPSIS

```
HELP:   Conversion from this CCSID [auto detection]
PATH:   DIFF.COMPARE.AUTO
TYPE:   STRING
SYNTAX: CCSID='str'/DEFAULT/ASCII/EBCDIC/BOMUTF/BOMUCS
```

DESCRIPTION The FROM code defines the encoding string or CCSID for the input text stream. If no CCSID is provided, the default handling (DEFAULT) is used. When reading, the input data's charset is auto-detected. The charsets below can currently be detected:

- ASCII (the finally used code page depends on the language code)

- EBCDIC (the finally used code page depends on the language code)
- UTF-8 == US-ASCII
- UTF-16LE == UCS-2LE
- UTF-16BE == UCS-2BE
- UTF-32LE == UCS-4LE
- UTF-32BE == UCS-4BE

If the data is encoded in ASCII or EBCDIC, the corresponding CCSID will be derived from the appropriate environment variable LANG. On non-ASCII systems, the ASCII-CCSID is derived from the language identifier (e.g. en, de , ..). The same holds true for non-EBCDIC systems if EBCDIC is detected. When ASCII is detected on an ASCII system, the CCSID from the LANG environment variable is used (e.g. de_DE.UTF-8 ⇒ UTF-8), if present. If the LANG variable contains no CCSID, then the language identifier is used (e.g. de_DE ⇒ ISO-8859-1). The same holds true for EBCDIC systems if EBCDIC is detected.

On modern platforms, the CCSID for a resource may be saved in the file system and can also be given by the FIO module. In general, however, it is recommended to set this parameter to its correct value. For default handling, the keywords DEFAULT, ASCII and EBCDIC as well as (for multibyte characters) the BOM keywords stated below can be used to help finding the correct CCSID on read operations.

Character conversions at write operation are also possible, but in this case, the BOM keywords are not supported. When writing, the FROM CCSID is usually known and does not need to be specified.

In order to use the BOM keywords, a BOM sign must be part of the input data. All unique BOM characters are supported in principle, but for UTF-16/USC-2 and UTF-32/UCS-4 the same BOM sign is used. You can define by yourself which family is used.

To get a list of supported encoding strings, please use the command below:

```
flcl INFO GET.ENCODINGS
```

Depending on the platform, this function lists all known encoding strings. Some encodings may not be supported. To determine the supported CCSIDs and corresponding encoding strings and charsets please use:

```
flcl INFO GET.CCSIDS
```

To define the CCSID, you can use one of the encoding strings or the decimal number for a CCSID displayed by the command above. Example:

```
CCSID='37'    will be equal to CCSID='IBM037'
CCSID='1208'  will be equal to CCSID='UTF-8'
```

A numeric CCSID must start with a decimal digit.

SELECTIONS

- `DEFAULT -Use default CCSID (auto-detect)`
- `ASCII -Use default ASCII CCSID (environment)`
- `EBCDIC -Use default EBCDIC CCSID (environment)`
- `BOMUTF -Determine the correct UTF CCSID from byte order mark (BOM)`
- `BOMUCS -Determine the correct UCS CCSID from byte order mark (BOM)`

4.7.188 PARAMETER *BOM*

SYNOPSIS

```
HELP:   Handle byte order change if get wrong BOM in the middle of data
PATH:   DIFF.COMPARE.AUTO
TYPE:   SWITCH
SYNTAX: BOM
```

DESCRIPTION If the handle byte order mark (BOM) keyword is specified, a byte order change is handled by the character conversion module. This means that a change of the byte order will not result in an error and the new byte order is used to read the next characters.

This can be useful for concatenated files with different byte orders. But in such case each file needs a BOM sign and the bit width must still be the same.

As default the handle BOM keyword is set to OFF.

4.7.189 PARAMETER *ENL2LF*

SYNOPSIS

```
HELP:   Replace EBCDIC new line (0x15) by line feed (0x25) on input
PATH:   DIFF.COMPARE.AUTO
TYPE:   SWITCH
SYNTAX: ENL2LF
```

DESCRIPTION This option only has an effect if the input CCSID is an EDCDIC charset. If enabled on ASCII systems, EBCDIC new line control characters (0x15) will be replaced by line feed characters (0x25) when character conversion takes place. When converting from an EBCDIC to a single-byte ASCII charset, then this option is enabled on EBCDIC system and can be disabled over this switch. This is required because a standard conform conversion converts 0x15 (EBCDIC NL) to 0x85 (ISO-8859-1/Latin1) instead of 0x0A. When converting to a UTF charset, 0x15 gets converted to 0xC285 (NEL) by default. Enabling this internal option will result in line feeds (0x0A) instead.

For the normal transfer of text data between ASCII and EBCDIC systems the internal adjustment of the translation table is done by FLAM automatically. Means normally you don't need to activate this switch, only in exceptional situations this option can be useful to achieve the opposite of the expected behavior.

This conversion is pre-calculated once at beginning of execution to reduce the required CPU time for the conversion.

4.7.190 OVERLAY *DECRYPT*

SYNOPSIS

```
HELP:   Decryption parameter [NONE]
PATH:   DIFF.COMPARE.AUTO
TYPE:   OVERLAY
SYNTAX: DECRYPT[{F4PWD()/F4KME()}...]
```

DESCRIPTION With the overlay decrypt you can define the decryption parameters for the decryption methods. For some decryption methods you must provide the passphrase, key label templates, email addresses or other information to use the correct key.

This overlay is an array which allows the definition of up to 32 decryption parameter sets. Each parameter set is used sequentially to attempt decryption of the data until one succeeds.

Here is an example with two passwords and an FKME which are tried for decryption. Both of the following notations are equivalent:

```
decr.f4pwd(pass='pwd1') decr.f4pwd(pass='pwd2') decr.f4kme()
decr[f4pwd(pass='pwd1') f4pwd(pass='pwd2') f4kme()]
```

See below for the decryption possibilities.

4.7.191 OBJECT *F4PWD*

SYNOPSIS

```
HELP:   Use FLAM password decryption
PATH:   DIFF.COMPARE.AUTO.DECRYPT
TYPE:   OBJECT
SYNTAX: F4PWD(PASSWORD/CRYPTOKEY='bin'/DEFAULT)
```

DESCRIPTION Uses a password for FLAM4 password-based decryption. If you provide no password, a hardcoded default password is used.

Passwords can be provided in four different ways:

```
 * In hexadecimal format (recommended): password=x'hex'
 * In ASCII format: password=a'ascii'
 * In EBCDIC format: password=e'ebcdic'
 * In local charset: password='system'
```

Note: Don't use the local charset variant if you wish to open the file on different platforms.

Important
Encrypting using the default password is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- `STRING:PASSWORD/CRYPTOKEY='bin'/DEFAULT -Passphrase to encrypt the FLAMFILE ['********']`
  - `DEFAULT -FLAM4 default password`

4.7.192 OBJECT *F4KME*

SYNOPSIS

```
HELP:   Use FLAM key management extension (FKME)
PATH:   DIFF.COMPARE.AUTO.DECRYPT
TYPE:   OBJECT
SYNTAX: F4KME(LIBRARY='str',FUNCTION='str',PARAMETER='str')
```

DESCRIPTION Activates FLAM4 decryption based on the FLAM4 key management extension (FKME). The FLAM data key is provided by an FKME module. The library name, function name and corresponding parameter list must be provided.

On mainframe systems, the load module name and the entry name must be identical, the load module must be in the STEPLIB concatenation and the library name is ignored. If only a function name is provided, then the default library name is "" on mainframes and "libfkme" on other platforms. If only a library name is provided, the default function name "FKMESTD0" is used.

If no FKME is provided (decr.f4kme()), then FKMESTD0 from the default library (see above) is used. The default FKMESTD0 returns a hardcoded key as encryption key identical to the default password when using decr.fl4pwd().

Important
Encrypting using the default FKME is strongly discouraged in production as this equals using a publicly known password! The data is merely obfuscated.

ARGUMENTS

- STRING:LIBRARY='str' -Library name for FKME ['libfkme']
- STRING:FUNCTION='str' -Function name of FKME ['FKMESTD0']
- STRING:PARAMETER='str' -Parameter for the call of FKME ['']

4.7.193 PARAMETER *LANG*

SYNOPSIS

```
HELP:   Redefine environment variable LANG for read operation
PATH:   DIFF.COMPARE.AUTO
TYPE:   STRING
SYNTAX: LANG='str'
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

4.7.194 PARAMETER *PLATFORM*

SYNOPSIS

```
HELP:   Redefine environment variable FL_PLATFORM for read operation
PATH:   DIFF.COMPARE.AUTO
TYPE:   STRING
SYNTAX: PLATFORM=WIN/UNX/ZOS/USS/VSE/BS2/MAC
```

DESCRIPTION The parameter LANG (mainly for character conversion) and the parameter platform (mainly for text formatting) can be used to overwrite corresponding environment variable LANG and FL_PLATFORM for the read and/or the write operation. This is useful to emulate another system at read or for write and use the automatism implemented by FLAM for character conversion and delimiter handling.

For example:

You have a file from a German Windows System on z/OS then it would be useful to set *LANG=DE_DE.CP1252* for the read operation and use the normal LANG environment variable defined in the FLCL configuration file for z/OS. If you want create a file for Windows on ZOS then the platform definition *WIN* for write ensure that as default the delimiter CRLF is used.

On the other side if you write a file on a UNIX system for USS on z/OS it would be useful to set the environment variable LANG for the write operation to *EN_US.IBM1047* and platform to *USS* and use the normal environment variable on the UNIX system for the read operation. In this case the file will be written in IBM1047 and the records are delimited with NL (0x15).

SELECTIONS

- WIN -Choose Windows as platform (CRLF)
- UNX -Choose Unix derivative as platform (LF)

- ZOS -Choose ZOS as platform (NL)
- USS -Choose USS of ZOS as platform (NL)
- VSE -Choose VSE as platform (NL)
- BS2 -Choose BS2000-OSD as platform (NL)
- MAC -Choose MAC as platform (CR)

4.7.195 OBJECT *HASH*

SYNOPSIS

```
HELP:   Generation or check of one way hash values of input data
PATH:   DIFF.COMPARE.AUTO
TYPE:   OBJECT
SYNTAX: HASH(ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24 ↩
    /CRC32/CRC32C/CRC40/CRC64,CHECK.{},FORMAT=GNU/BSD,OUTPUT='str'/STDOUT/STDERR)
```

DESCRIPTION The hash component can be used to generate and check hash values (checksums). This can be useful to verify data integrity of files after transmission or on a storage device.

If you do not specify the output filename, the output of this component is written to the log.

For the output file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   OUTPUT=STDERR                          ; is SYSPRINT on z/OS
   OUTPUT='~.MYCHECK(HASH1)'              ; PDS on z/OS
   OUTPUT='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   OUTPUT='~/mycheck3.txt'                ; Unix path name
   OUTPUT='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

By using the *check* overlay, data can be verified against a previously computed hash value that can either be passed directly or from an input file. The result is written to output.

Passing an empty string or filename in the *check* overlay implies only calculation of the hash without verification.

ARGUMENTS

- NUMBER:ALGO=MD5/RIPEMD128/RIPEMD160/SHA1/SHA224/SHA256/SHA384/SHA512/CRC8/CRC16/CRC24/CRC32/CRC32C/CRC40/CRC64 -Hash algorithm [SHA1]
  - MD5 -Message-digest algorithm 5 (128 bit)
  - RIPEMD128 -RACE Integrity Primitives Evaluation Message Digest (128 bit)
  - RIPEMD160 -RACE Integrity Primitives Evaluation Message Digest (160 bit)
  - SHA1 -Secure hash algorithm 1 (160 bit)
  - SHA224 -Secure hash algorithm 2 (224 bit)
  - SHA256 -Secure hash algorithm 2 (256 bit)
  - SHA384 -Secure hash algorithm 2 (384 bit)
  - SHA512 -Secure hash algorithm 2 (512 bit)
  - CRC8 -Cyclic redundancy check (8 bit)
  - CRC16 -Cyclic redundancy check (16 bit)
  - CRC24 -Cyclic redundancy check (24 bit)
  - CRC32 -Cyclic redundancy check (32 bit)
  - CRC32C -Cyclic redundancy check (32 bit) Castagnoli variant

  - CRC40 -Cyclic redundancy check (40 bit)
  - CRC64 -Cyclic redundancy check (64 bit)
- NUMBER:FORMAT=GNU/BSD -Define printout format [GNU]
  - GNU -Write output in GNU style
  - BSD -Write output in BSD style
- STRING:OUTPUT='str'/STDOUT/STDERR -Name of the output file [LOG]
  - STDOUT -Write output to stdout
  - STDERR -Write output to stderr

4.7.196 OVERLAY *CHECK*

SYNOPSIS

```
HELP:   Parameter for checksum verification
PATH:   DIFF.COMPARE.AUTO.HASH
TYPE:   OVERLAY
SYNTAX: CHECK.{VALUE='str'/FILE='str'/STREAM/DUMMY}
```

DESCRIPTION The *check* parameter defines the expected hash (checksum).

The hash can be specified directly or read from a file. For the check file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   FILE=STREAM                          ; STDIN is SYSIN on z/OS
   FILE='~.MYCHECK(HASH1)'              ; PDS on z/OS
   FILE='<SYSUID>.MYCHECK2'             ; PS dataset on z/OS
   FILE='~/mycheck3.txt'                ; Unix path name
   FILE='<HOME>\hashs\mycheck4.txt'     ; Windows path name
```

Currently, chescksum files can be in standard GNU format:

```
   hex_hash_checksum   path_to_orignal_file
```

or BSD style:

```
   ALGO(path_to_orignal_file) = hex_hash_checksum
```

ARGUMENTS

- STRING:VALUE='str' -Checksum as hex-string
- STRING:FILE='str'/STREAM/DUMMY -File which contains Checksum as hex-string
  - STREAM -Read from stdin or write to stdout
  - DUMMY -Read EOF or write nothing

4.7.197 PARAMETER *CHECK*

SYNOPSIS

```
HELP:   Activate additional optional verifications [FALSE]
PATH:   DIFF.COMPARE.AUTO
TYPE:   SWITCH
SYNTAX: CHECK
```

DESCRIPTION The CHECK switch of CONV.READ activates the additional verifications below:

- Check the CRC checksum that is appended to ASCII-armored data (if available)
- Check the ASCII armor trailer against the header when decoding ASCII armor

In the future, there will be more optional verifications, which can be activated through this switch.

4.7.198 PARAMETER *FILE*

SYNOPSIS

```
HELP:   Abbreviation for read.auto(file=...)
PATH:   DIFF.COMPARE
TYPE:   STRING
SYNTAX: FILE='str'/STREAM/DUMMY
```

DESCRIPTION `read.file='name.txt'` is an abbreviation for `read.auto(file='name.txt')` without any further parameters. Please have a look at read.auto() for more information.

`read.file` use all data format detection capabilities of FLAM to read transparent byte streams or records form a data source, which can be an encoded (e.g. Base64), encrypted (e.g. PGP) and/or compressed (GZIP/BZIP2/XZ) file, a FLAMFILE, other archive or a normal clear data set.

The data stream after I/O and after each conversion can contain several kinds of 4 byte length field formats. This is also detected and the data is interpreted as blocks containing various records for the next layer.

SELECTIONS

- `STREAM -Read from stdin or write to stdout`
- `DUMMY -Read EOF or write nothing`

4.7.199 OVERLAY *LOGGING*

SYNOPSIS

```
HELP:   Select target for logging
PATH:   DIFF
TYPE:   OVERLAY
SYNTAX: LOGGING.{STREAM()/FILE()/SYSTEM()}
```

DESCRIPTION The log destination parameter defines where the log messages will be written.

Each log message starts with the current time and date.

All the possible destinations allow to specify a custom identification string which will be printed after time and date of each message. This is done with the IDENT argument.

If no IDENT specified the default will be *owner.program.command*

If no log destination defined stream will be used as default.

4.7.200 OBJECT *STREAM*

SYNOPSIS

```
HELP:   Log to a stream
PATH:   DIFF.LOGGING
TYPE:   OBJECT
SYNTAX: STREAM(FORMAT=STANDARD/DIALOG,IDENT='str',STDOUT)
```

DESCRIPTION The STREAM object allows the log messages to be written to the standard output or the standard error output. Standard error output is used if STDOUT is not specified.

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- SWITCH:STDOUT -Log messages to STDOUT instead of STDERR

4.7.201 OBJECT *FILE*

SYNOPSIS

```
HELP:   Log to a flat file
PATH:   DIFF.LOGGING
TYPE:   OBJECT
SYNTAX: FILE(FORMAT=STANDARD/DIALOG,IDENT='str',NAME='str')
```

DESCRIPTION The flat file option allows the log messages to be written to a normal file. The name of the file must be specified for this destination. The log messages will be appended to this file if it already exists.

For the log file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   NAME='~.MYLOG(LOG1)'                  ; PDS on z/OS
   NAME='<SYSUID>.MYLOG02'               ; PS dataset on z/OS
   NAME='~/mylog3.txt'                   ; Unix path name
   NAME='<HOME>\logs\mylog4.txt'         ; Windows path name
```

ATTENTION: Parallel process cannot share the same log file

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- STRING:NAME='str' -Filename for appending log messages

4.7.202 OBJECT *SYSTEM*

SYNOPSIS

```
HELP:   Use system logging facility
PATH:   DIFF.LOGGING
TYPE:   OBJECT
SYNTAX: SYSTEM(IDENT='str')
```

DESCRIPTION The SYSTEM destination allows the log messages to be written to the logging system of the operating system.

On Unix and z/OS® systems the syslog() call is used. On Windows® the event logging facility is used.

```
NOTE: On Windows(R) this requires the registry key entry:
      HKEY_LOCAL_MACHINE
        SYSTEM
          CurrentControlSet
            Services
              EventLog
                Application
                  FLM
```

This registry key will be created by using the DLL flevent.dll

ARGUMENTS

- `STRING:IDENT='str' -Identifier used to identify log entries`

4.7.203 OBJECT *MESSAGE*

SYNOPSIS

```
HELP:   Select type of messages to log
PATH:   DIFF
TYPE:   OBJECT
SYNTAX: MESSAGE(NONE,ERROR,ERRTRACE,WARNING,NOTICE,DEBUG,ABOUT,VERSION,INPUT,PARAMETER, ↩
    STATISTIC,INFO,LICID,LICENSE,PROGRESS,ALL,DEFAULT,MINIMAL,ERRONLY)
```

DESCRIPTION The log message specification is a classification of the messages to write to the selected log destination. Each type is selectable individually by adding its argument inside this object.

For added convenience, the special types NON, ALL, MINIMAL select the proper message types with just one argument.

If no message specification is given, a DEFAULT is used which writes out messages of these types:

- ERROR
- ERRTRACE
- WARNING
- NOTICE
- PARAMETER
- STATISTIC
- INFO
- LICID

ARGUMENTS

- `SWITCH:NONE -No messages will be logged`
- `SWITCH:ERROR -Error messages will be logged`
- `SWITCH:ERRTRACE -Error trace will be logged`
- `SWITCH:WARNING -Warnings will be logged`
- `SWITCH:NOTICE -Notices will be logged`
- `SWITCH:DEBUG -Debug information will be logged`

- SWITCH:ABOUT -About information will be logged
- SWITCH:VERSION -Version information will be logged
- SWITCH:INPUT -Input data will be logged (dangerous for passwords)
- SWITCH:PARAMETER -Parsed parameters will be logged
- SWITCH:STATISTIC -Statistics will be logged
- SWITCH:INFO -Requested information will be logged
- SWITCH:LICID -License ID will be logged
- SWITCH:LICENSE -License summary will be logged
- SWITCH:PROGRESS -Progress bar will be logged
- SWITCH:ALL -All messages are logged
- SWITCH:DEFAULT -Relevant messages are logged
- SWITCH:MINIMAL -Only important messages are logged
- SWITCH:ERRONLY -Only errors and error trace are logged

4.8 COMMAND *UTIL*

SYNOPSIS

```
HELP:   Executes simply functions
PATH:   LIM.flcl
TYPE:   OBJECT
SYNTAX: :> flcl UTIL(RUN.{},DIR(),LOG())
```

DESCRIPTION The UTIL command executes a function on a wildcard-based list of files. This command allows to run simple system functions like deleting or listing files that match a provided pattern which may contain wildcards. This is mainly useful on mainframe systems, for example, to find all libraries containing a certain member or to delete any kind of file under a certain qualifier.

The UTIL command prints only minimal log output. For a more verbose output (e.g. notices), please use the log object (see examples below).

To get syntax information, please use:

```
flcl SYNTAX UTIL
```

To get help for a parameter, please use:

```
flcl HELP UTIL.parameter[.parameter[...]]
```

To read the manual page for a parameter, please use:

```
flcl MANPAGE UTIL.parameter[.parameter[...]]
   or
flcl HELP UTIL.parameter[.parameter[...]] MAN
```

To generate the user manual for a command, please use:

```
flcl GENDOCU UTIL=filename
```

Parameters can be defined by the command line (directly or per file) or by properties read from the corresponding property file.

EXAMPLES

```
   flcl UTIL run.list='**(CSFMOD01)'   # list all libs containing this member
   flcl UTIL run.remove='~.TEST.**'    # deletes all test files under SYSUID
   flcl UTIL run.remove='*.txt' log(message(default)) # activate default logging
```

4.8.1 OVERLAY *RUN*

SYNOPSIS

```
HELP:   Specify what you want to do
PATH:   UTIL
TYPE:   OVERLAY
SYNTAX: RUN.{LIST='str'/REMOVE='str'}
```

DESCRIPTION With the overlay RUN you choose which utility function to use. Be careful with the remove operation for files. Please use the list function first to see which files will get deleted when using wildcards.

All standard outputs (mainly the file list) of the utility functions are written to STDOUT.

You can find the currently supported functions below. Don't hesitate to request new functions that are useful with wildcard patterns.

ARGUMENTS

- `STRING:LIST='str' -List files`
- `STRING:REMOVE='str' -Remove files`

4.8.2 OBJECT *DIR*

SYNOPSIS

```
HELP:   Defines how to handle directories and various file types if wildcards used
PATH:   UTIL
TYPE:   OBJECT
SYNTAX: DIR(LINK,ALIAS,HIDDEN,RECURSIVE,ARCHIVE,ONEFILESYSTEM,TAPE)
```

DESCRIPTION This object defines multiple switches that control how input files are found. This is especially important when using wildcards in input filenames.

There are switches to enable recursion into subdirectories, following symbolic links, finding hidden files, recursion into archive files, include files on tapes, resolving aliases and enforcing to not leave the filesystem.

If the recursive flag is set, the filename portion of the input file string is searched for recursively. If a directory path containing wildcards is present, the filename portion is searched for recursively in all paths that match the directory path pattern, e.g. "**/path/**.txt" matches my/path/hugo.txt, another/path/hugo.txt as well as my/path/subdir/hugo.txt. Without the recursive flag, the last file would not be found.

If the archive switch is enabled, FLAMFILES, TARBALLS, ZIP files and other kinds of supported archives are searched. All the matching members are extracted. In other words, archives are interpreted like subdirectories if the flag is enabled.

If the tape switch is enabled, then datasets stored on tapes are searched. If the name of such a file matches the pattern, then it might occur that console input and a manual intervention is required to retrieve the respective dataset(s).

If the link switch is enabled, symbolic links are followed (otherwise they are ignored). Links are mainly supported on modern file systems. For dataset aliases on mainframe systems, see the alias flag.

With the alias switch enabled, datasets names which are aliases of the real dataset are resolved to their actual names (otherwise they are ignored). Dataset aliases only exist on mainframe systems. This flag has no effect on other platforms.

If a transparent read method (e.g. read.auto()) is used, then the archive flag is enabled by default and cannot be disabled.

ARGUMENTS

- SWITCH:LINK –Include symbolic links
- SWITCH:ALIAS –Resolve data set aliases
- SWITCH:HIDDEN –Include hidden files
- SWITCH:RECURSIVE –Include sub directories
- SWITCH:ARCHIVE –Include archives (FLAMFILEs, ZIP, ...)
- SWITCH:ONEFILESYSTEM –Don't cross file system boundaries
- SWITCH:TAPE –Include files deposited on tapes

4.8.3 OBJECT *LOG*

SYNOPSIS

```
HELP:   Logging Parameter
PATH:   UTIL
TYPE:   OBJECT
SYNTAX: LOG(MESSAGE(),DESTINATION.{})
```

DESCRIPTION The LOG parameter allows flexible adjustment for the logging of messages. It is possible to specify the destination as well as which kind of message will be written.

For more information please have a look at the man pages for each parameter of LOG.

4.8.4 OBJECT *MESSAGE*

SYNOPSIS

```
HELP:   Select type of messages to log
PATH:   UTIL.LOG
TYPE:   OBJECT
SYNTAX: MESSAGE(NONE,ERROR,ERRTRACE,WARNING,NOTICE,DEBUG,ABOUT,VERSION,INPUT,PARAMETER, ↩
    STATISTIC,INFO,LICID,LICENSE,PROGRESS,ALL,DEFAULT,MINIMAL,ERRONLY)
```

DESCRIPTION The log message specification is a classification of the messages to write to the selected log destination. Each type is selectable individually by adding its argument inside this object.

For added convenience, the special types NON, ALL, MINIMAL select the proper message types with just one argument.

If no message specification is given, a DEFAULT is used which writes out messages of these types:

- ERROR
- ERRTRACE
- WARNING
- NOTICE
- PARAMETER
- STATISTIC
- INFO
- LICID

ARGUMENTS

- SWITCH:NONE -No messages will be logged
- SWITCH:ERROR -Error messages will be logged
- SWITCH:ERRTRACE -Error trace will be logged
- SWITCH:WARNING -Warnings will be logged
- SWITCH:NOTICE -Notices will be logged
- SWITCH:DEBUG -Debug information will be logged
- SWITCH:ABOUT -About information will be logged
- SWITCH:VERSION -Version information will be logged
- SWITCH:INPUT -Input data will be logged (dangerous for passwords)
- SWITCH:PARAMETER -Parsed parameters will be logged
- SWITCH:STATISTIC -Statistics will be logged
- SWITCH:INFO -Requested information will be logged
- SWITCH:LICID -License ID will be logged
- SWITCH:LICENSE -License summary will be logged
- SWITCH:PROGRESS -Progress bar will be logged
- SWITCH:ALL -All messages are logged
- SWITCH:DEFAULT -Relevant messages are logged
- SWITCH:MINIMAL -Only important messages are logged
- SWITCH:ERRONLY -Only errors and error trace are logged

4.8.5 OVERLAY *DESTINATION*

SYNOPSIS

```
HELP:   Destination for log messages
PATH:   UTIL.LOG
TYPE:   OVERLAY
SYNTAX: DESTINATION.{STREAM()/FILE()/SYSTEM()}
```

DESCRIPTION The log destination parameter defines where the log messages will be written.

Each log message starts with the current time and date.

All the possible destinations allow to specify a custom identification string which will be printed after time and date of each message. This is done with the IDENT argument.

If no IDENT specified the default will be *owner.program.command*

If no log destination defined stream will be used as default.

4.8.6 OBJECT *STREAM*

SYNOPSIS

```
HELP:   Log to a stream
PATH:   UTIL.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: STREAM(FORMAT=STANDARD/DIALOG,IDENT='str',STDOUT)
```

DESCRIPTION The STREAM object allows the log messages to be written to the standard output or the standard error output. Standard error output is used if STDOUT is not specified.

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- SWITCH:STDOUT -Log messages to STDOUT instead of STDERR

4.8.7 OBJECT *FILE*

SYNOPSIS

```
HELP:   Log to a flat file
PATH:   UTIL.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: FILE(FORMAT=STANDARD/DIALOG,IDENT='str',NAME='str')
```

DESCRIPTION The flat file option allows the log messages to be written to a normal file. The name of the file must be specified for this destination. The log messages will be appended to this file if it already exists.

For the log file, all rules and replacements are valid which are described under "FILENAME HANDLING" above. For example:

```
   NAME='~.MYLOG(LOG1)'                  ; PDS on z/OS
   NAME='<SYSUID>.MYLOG02'               ; PS dataset on z/OS
   NAME='~/mylog3.txt'                   ; Unix path name
   NAME='<HOME>\logs\mylog4.txt'         ; Windows path name
```

ATTENTION: Parallel process cannot share the same log file

ARGUMENTS

- NUMBER:FORMAT=STANDARD/DIALOG -Format of log message [STANDARD]
  - STANDARD -Standard logging format (timestamp ident *level* message)
  - DIALOG -For dialog processing (ident *level* message)
- STRING:IDENT='str' -Identifier used to identify log entries
- STRING:NAME='str' -Filename for appending log messages

4.8.8 OBJECT *SYSTEM*

SYNOPSIS

```
HELP:   Use system logging facility
PATH:   UTIL.LOG.DESTINATION
TYPE:   OBJECT
SYNTAX: SYSTEM(IDENT='str')
```

DESCRIPTION The SYSTEM destination allows the log messages to be written to the logging system of the operating system.

On Unix and z/OS® systems the syslog() call is used. On Windows® the event logging facility is used.

```
NOTE: On Windows(R) this requires the registry key entry:
      HKEY_LOCAL_MACHINE
        SYSTEM
          CurrentControlSet
            Services
              EventLog
                Application
                  FLM
```

This registry key will be created by using the DLL flevent.dll

ARGUMENTS

- `STRING:IDENT='str' -Identifier used to identify log entries`

Chapter 5

Available built-in functions

Numerous built-in functions are available to obtain extensive help about syntax, errors, commands and functions of a program with help messages, manual pages and documentation.

Other built-in functions can be used for owner, property, environment and trace management. All built-in functions have the purpose to simplify handling of powerful and sometimes complex commands.

5.1 FUNCTION *SYNTAX*

SYNOPSIS

```
HELP:   Provides the syntax for each command
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl SYNTAX [command[.path] [DEPTH1 | ... | DEPTH9 | ALL]]
```

DESCRIPTION The syntax function shows how the arguments can be set on the command line. Depending on depth the syntax is printed in one line or in structured form. A structured print out means that you have a question or exclamation mark before the argument.

- question mark - optional parameter
- exclamation mark - required parameter

The **path** is a dotted keyword list in accordance with the syntax of the corresponding command.

```
command.para1.para2.para3
```

OPTIONS

```
DEPTH1 / DEPTH2 / ... / DEPTH9
```

Specifies the depth of the output. Default is DEPTH1. If the depth is greater than 1 the structured print out is used.

```
ALL
```

Shows complete syntax for the specified path, not just DEPTH1. Structured print out is used.

EXAMPLES

```
:> FLCL SYNTAX
:> FLCL SYNTAX command
:> FLCL SYNTAX command ALL
:> FLCL SYNTAX command.para
:> FLCL SYNTAX command.para ALL
```

5.2 FUNCTION *HELP*

SYNOPSIS

```
HELP:   Provides quick help for arguments
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl HELP [command[.path] [DEPTH1 | ... | DEPTH9 | ALL]] [MAN]
```

DESCRIPTION The help function can provide extensive information about each parameter which is adjustable on the command line. With the help functionality you get a structured list of arguments:

- keyword - the keyword for the argument
- (TYPE: ...) - the type of this argument
- help message - the short help text
- (PROPERTY: [...]) - the current hard coded property value (optional)

The keyword can be abbreviated, but it must be unique for this list. The required letters are upper case if case sensitive mode is used. With the case sensitive mode an additional line is required to mark the appropriate letter.

The **path** is a dotted keyword list in accordance with the syntax of the corresponding command.

```
command.para1.para2.para3
```

Help can also be used to show the corresponding detailed description of the manual pages with keyword **MAN**.

OPTIONS

```
DEPTH1 / DEPTH2 / ... / DEPTH9
```

Specifies the depth of the output. Default is DEPTH1.

```
ALL
```

Shows complete help below specified path, not just DEPTH1 and includes all aliases defined for a certain argument.

```
MAN
```

Shows additionally the detailed description for this program, command or parameter and the next level of arguments including the aliases.

EXAMPLES

```
   :> FLCL HELP
   :> FLCL HELP MAN
   :> FLCL HELP command
   :> FLCL HELP command MAN
   :> FLCL HELP command ALL
   :> FLCL HELP command.para
   :> FLCL HELP command.para MAN
   :> FLCL HELP command.para ALL
```

5.3 FUNCTION *MANPAGE*

SYNOPSIS

```
HELP:   Provides manual pages (detailed help)
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl MANPAGE [function | command[.path][=filename]] | [filename]
```

DESCRIPTION Shows the manual pages of the program, all commands and built-in functions and if available for each argument up to selections/constant definitions.

This function prints the corresponding section from the user manual on the screen (doctype: **book**). If a filename is given then the doctype **manpage** of ASCIIDOC is written to the file.

Do not use space characters between the key, assignment character (=) and filename.

The files in ASCIIDOC format can be used to generate manpages with the tool ASCIIDOC.

EXAMPLES

```
   :> FLCL MANPAGE
   :> FLCL MANPAGE filename
   :> FLCL MANPAGE command
   :> FLCL MANPAGE command=filename
   :> FLCL MANPAGE function
   :> FLCL MANPAGE function=filename
   :> FLCL MANPAGE command.para
   :> FLCL MANPAGE command.para=filename
```

5.4 FUNCTION *GENDOCU*

SYNOPSIS

```
HELP:   Generates auxiliary documentation
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl GENDOCU [command[.path]=]filename [NONBR]
```

DESCRIPTION The complete user manual is generated when only the filename is given. If a command, or command path, is specified the corresponding section of the user manual is written to a file. The generated files are in ASCIIDOC format, to be readable as normal text document on all platforms and easily convertible into HTML, PDF or other formats with ASCIIDOC.

The numbering will vary in places where the complete manual uses a bullet list for arguments and constants, when no detailed description is available and some of these arguments are required for the path.

The numbering can be disabled by setting the optional keyword NONBR. This is useful if numbering is generated by a post processing utility (for example *pdflatex*).

Do not use space characters between the key, assignment character (=) and filename.

EXAMPLES

```
   :> FLCL GENDOCU flcl.manual.txt NONBR
   :> FLCL GENDOCU command=command.manual.txt
   :> FLCL GENDOCU command.argument=argument.manual.txt
```

5.5 FUNCTION *GENPROP*

SYNOPSIS

```
HELP:   Generates a property file
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl GENPROP [command=]filename
```

DESCRIPTION A property file is generated for all commands if only a filename is provided. If a command is specified, a property file for this specific command is created. Property files can be used to define default settings for commands. For each parameter, a property entry is written to a text file. The text file contains comments to help in editing the properties. A property file can be activated with the built-in function SETPROP.

Property files will be activated by their owner. A property file overrules hard coded properties and properties defined over environment variables. A specific property file overrules the general property file. At time of generation all current default settings are written to the property file.

We recommend to work only with command specific property files, because the built-in function CHGPROP can be used to generate, update and activate a property file for a certain command.

Do not use space characters between the key, assignment character (=) and filename.

To optimize the program performance, one can remove all unnecessary properties.

See Appendix C for the current property file content.

EXAMPLES

```
   :> FLCL GENPROP owner.general.properties
   :> FLCL GENPROP command=owner.command.properties
```

5.6 FUNCTION *SETPROP*

SYNOPSIS

```
HELP:   Activate a property file
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl SETPROP [command=]filename
```

DESCRIPTION The file name is set as current global property file when no command is specified. The file name is set as current local property file when a command is given. The property file can be generated with GENPROP and will be activated for the current owner.

After generation you can activate the property file, but there will be no effect until you change the settings inside the property file, because the current default settings are written.

The file name can contain replacement/mapping rules (<HOME>/<USER>).

Do not use space characters between the key, assignment character (=) and filename.

We recommend to work only with command specific property files, because the built-in function CHGPROP can be used to generate, update and activate a property file for a certain command.

See Appendix C for the current property file content.

EXAMPLES

```
   :> FLCL SETPROP general.properties
   :> FLCL SETPROP command=command.properties
```

5.7 FUNCTION *CHGPROP*

SYNOPSIS

```
HELP:   Change a property value in the currently active property file
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl CHGPROP command [path[=value]]*
```

DESCRIPTION The built-in function CHGPROP updates property values in the currently active (see SETPROP) property file for a dedicated command. If no property file is activated, a property file is generated (see GENPROP), updated with the provided property list and activated (see SETPROP).

The first argument of CHGPROP is the command name. All other arguments are key-value pairs of properties that should be updated. The key is internally prefixed with the root (owner.program.command.). The values are enclosed in double quotation marks. Multiple properties of a command can be updated at once by separating them with a space character (please do not use spaces between key, assignment character (=) and the value). This set of properties is parsed by the CLP. If no property is provided, the property file for the command is either parsed or generated (if it does not exist) and then activated without updating properties. This behavior can be used to generate and activate command specific property files for the different commands.

The built-in function reads and parses the currently active property file. Then it parses the provided property list and writes the updated property file back to disk. The currently active property file could be a global, not command specific property file. In such a case, a new command specific property file is generated and activated. Properties from the global property file are copied into the newly generated one.

If no property file is defined (see SETPROP) that corresponds to the owner, program and command, a property file is generated. The filename formats below are used to read or write property files from/to the home directory:

On non-mainframe systems (WINDOWS, UNIX, MAC): ".owner.program.command.properties"

On mainframe systems (ZOS, VSE, BS200, ...): "<SYSUID>.OWNER.PROGRAM.COMMAND.PROPS"

If the default filenames and/or path are not sufficient, you can change the filename of each property file by setting environment variables using the following naming convention:

```
OWNER_PROGRAM_COMMAND_PROPERTY_FILENAME
```

If no home directory is defined, then the current working directory is used as default directory to save the property files. On mainframes SYSUID is used for the first level qualifier, to represent the "home directory".

Property files are managed per owner: This means updates are only done for the current owner.

To delete a property you can simply specify the path without the root and with out a sign.

See Appendix C for the current property file content.

EXAMPLES

```
   :> FLCL CHGPROP command overlay.object.argument=value
   :> FLCL CHGPROP command overlay.object.argument
   :> FLCL CHGPROP command object.argument=value1 argument=value2
```

5.8 FUNCTION *DELPROP*

SYNOPSIS

```
HELP:   Remove a property file from configuration
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl DELPROP [command]
```

DESCRIPTION Without a command the current global property file is deleted from the configuration data. A command specific property file can be deleted from the configuration data when the command name is given.

Property files are managed per owner, this means delete is only done for the current owner.

We recommend to work only with command specific property files.

See Appendix C for the current property file content.

EXAMPLES

```
:> FLCL DELPROP
:> FLCL DELPROP command
```

5.9 FUNCTION *GETPROP*

SYNOPSIS

```
HELP:   Show current properties
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl GETPROP [command[.path] [DEPTH1 | ... | DEPTH9 | DEPALL | DEFALL]]
```

DESCRIPTION All or certain properties can be shown for a dedicated path. The **path** is a dotted keyword list according to the syntax of the corresponding command.

```
command.para1.para2.para3
```

A help message for a certain parameter will also show the assigned property value.

See Appendix C for the current property file content.

OPTIONS

```
DEPTH1 / DEPTH2 / ... / DEPTH9
```

Specifies the depth of the output. Default is DEPTH1.

```
DEPALL
```

Shows all properties to the specified path, not just DEPTH1.

```
DEFALL
```

Shows all defined properties to the specified path.

EXAMPLES

```
:> FLCL GETPROP
:> FLCL GETPROP command
:> FLCL GETPROP command DEFALL
:> FLCL GETPROP command.para
:> FLCL GETPROP command.para DEPALL
```

5.10 FUNCTION *SETOWNER*

SYNOPSIS

```
HELP:   Defines the current owner
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl SETOWNER name
```

DESCRIPTION The current owner id (prefix or separation) can be changed. The owner concept is very powerful to separate different property settings. You can run the same command with different properties simply by switching the owner. The owner is a general solution to separate all settings and can be used for environment, client and other purposes.

If not already defined, the owner is written to the environment table under the key word OWNERID. This environment variable can for example used as replacement in file names (~/<OWNERID>.dat) or to refer a signature key.

EXAMPLES

```
   :> FLCL SETOWNER com.company
```

5.11 FUNCTION *GETOWNER*

SYNOPSIS

```
HELP:   Show current owner setting
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl GETOWNER
```

DESCRIPTION The current owner id (prefix for separation) will be shown. The owner concept allows to manage different configurations as a logical separation of what is done with this id.

EXAMPLES

```
   :> FLCL GETOWNER
```

5.12 FUNCTION *SETENV*

SYNOPSIS

```
HELP:   Set an environment variable
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl SETENV variable=value
```

DESCRIPTION This function allows to set environment variables within the configuration file for a certain owner. All defined environment variables are placed in the current environment table of the process before a command is executed.

Do not use space characters between the key, assignment character (=) and value.

Attention: The environment variables will be temporary overwritten

EXAMPLES

```
   :> FLCL SETENV LANG=DE_DE.IBM-1141
   :> FLCL SETENV LANG=DE_DE.UTF-8
```

5.13 FUNCTION *GETENV*

SYNOPSIS

```
HELP:   Show the environment variables
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl GETENV
```

DESCRIPTION This function lists all environment variables which are set within the configuration file for a certain owner. All shown environment variables are placed in the environment table of the process.

Attention: The environment variables will be temporary overwritten

EXAMPLES

```
   :> FLCL GETENV
```

5.14 FUNCTION *DELENV*

SYNOPSIS

```
HELP:   Delete an environment variable
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl DELENV variable
```

DESCRIPTION This function will delete environment variables from the configuration file for a certain owner.

EXAMPLES

```
   :> FLCL DELENV LANG
```

5.15 FUNCTION *TRACE*

SYNOPSIS

```
HELP:   Manage trace capabilities
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl TRACE ON | OFF | FILE=filename
```

DESCRIPTION Tracing for command line parser and commands can be enabled or disabled. A trace is only required to diagnose errors. In such a case a trace file must be defined first, then the trace can be activated, the erroneous command executed and the trace file sent to the support team.

If no trace file is defined, stdout/stderr is used as default. To prevent tracing on the screen, please define a trace file first before tracing is switched on.

Do not use space characters between the key, assignment character (=) and filename.

EXAMPLES

```
   :> FLCL TRACE FILE=filename
   :> FLCL TRACE ON
   :> FLCL TRACE OFF
```

5.16 FUNCTION *CONFIG*

SYNOPSIS

```
HELP:   Shows or clear all the current configuration settings
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl CONFIG [CLEAR]
```

DESCRIPTION The actual configuration list with owner id, property file names trace setting, environment variables and more are shown. With the option CLEAR the configuration data including the corresponding file will be deleted.

The configuration file name can be defined by the environment variable:

```
FLCL_CONFIG_FILE=config.file.name
```

This can be used to work with a global configuration file. If this location not defined, then the configuration file are stored on distributed systems in the current home directory under the file name:

```
.flcl.config
```

If no home directory defined for the current user (this case is very improbable), then the working directory is used.

To read the configuration file and the environment variable above is not defined, CLE looks first in the working and then in the home directory. This enables you to prefer a property file, by coping it to the current working directory.

On mainframes as default the data set name *SYSUID.FLCL.CONFIG* is used. SYSUID as high level qualifier is a kind of replacement for the current home directory on distributed system in this case.

EXAMPLES

```
   :> FLCL CONFIG
   :> FLCL CONFIG CLEAR
```

5.17 FUNCTION *GRAMMAR*

SYNOPSIS

```
HELP:   Shows the grammar for commands and properties
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl GRAMMAR
```

DESCRIPTION The grammar used for interpreting the command line and property files is shown. This is helpful for understanding how the commands and property files are interpreted by the command line processor.

See Appendix B for an explanation and the output of this command.

EXAMPLES

```
   :> FLCL GRAMMAR
```

5.18 FUNCTION *LEXEM*

SYNOPSIS

```
HELP:   Shows the regular expressions accepted in a command
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl LEXEM
```

DESCRIPTION The regular expression used for the tokens of the command syntax will be shown. This could be helpful in understanding how the commands and property files are interpreted by the command line processor. Moreover it will be shown how to define a number, a string, a keyword and other values inside a command.

See Appendix A for an explanation and the output of this command.

EXAMPLES

```
   :> FLCL LEXEM
```

5.19 FUNCTION *LICENSE*

SYNOPSIS

```
HELP:   List license information for the program
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl LICENSE
```

DESCRIPTION The license string of the software product is shown. This string is provided to the command line executer (CLE) by the calling program and should reflect the current license text for this program.

EXAMPLES

```
   :> FLCL LICENSE
```

5.20 FUNCTION *VERSION*

SYNOPSIS

```
HELP:   List version information for the program
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl VERSION
```

DESCRIPTION The version of the software product is shown. This string is provided to the command line executer (CLE) by the calling program and should reflect the current version information for this program.

See Appendix F for the output of this command.

EXAMPLES

```
   :> FLCL VERSION
```

5.21 FUNCTION *ABOUT*

SYNOPSIS

```
HELP:   Show information about the program
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl ABOUT
```

DESCRIPTION Shows information about the software product like version, copyright, external libraries, license and other attributes. This string is provided to the command line executer (CLE) by the calling program and should reflect the current about information for this program (flcl).

See Appendix G for the output of this command.

EXAMPLES

```
   :> FLCL ABOUT
```

5.22 FUNCTION *ERRORS*

SYNOPSIS

```
HELP:   Show information about return and reason codes of the program
PATH:   LIM.flcl
TYPE:   BUILT-IN FUNCTION
SYNTAX: :> flcl ERRORS
```

DESCRIPTION This built-in function prints explanations for the return/condition/exit codes of the executable and, if available, the reason codes of executed commands.

The command line executer (CLE) initialized the CLP structure. Calls the command line processor (CLP). Calls the mapping function, which converts the CLP structure in a corresponding command structure. Then it runs the command and as last step it calls the finish function with the command structure, to close all files and free all the allocated memory. If one of these steps fails, then a corresponding return (condition/exit) code is given back from the executable. Only the command execution can result in a error (return code 8) or in a warning (return code 4). The command it self must return a reason code for such a warning or error. Means the return (condition/exit) code can be used to control the batch processing. The reason code gives more information about the error or warning.

See Appendix D and Appendix E for an explanation and the output of this command.

EXAMPLES

```
   :> FLCL ERRORS
```

Chapter 6

GLOSSARY

ASCIIDOC A text document format to write notes, documentation, articles, books, ebooks, slideshows, web pages, man pages and blogs. ASCIIDOC files can be converted into HTML, PDF, EPUB or man pages. ASCIIDOC is highly configurable, as the syntax and the backend output markups can be customized and extended by the user. It is licensed under the terms of the GNU General Public License version 2. The complete manual or parts of it can be built from the ASCIIDOC files and translated to current formats like HTML or PDF. The CLP uses the command GENDOCU to generate ASCIIDOC files. You just have to specify the file name. For ASCIIDOC see: ASCIIDOC

BATCH Batch processing is execution of a series of programs ("jobs") on a computer without manual intervention.

BUILT-IN FUNCTIONS The command line processor CLP provides a number of functions built- in that are available to obtain extensive syntax information, help messages, manual pages and documentation. There are additional built- in functions that can be used for owner, property and trace manage- ment.

CHG Change

CHANGING OF PROPERTIES With the built-in function CHGPROP, you can update a property value in the currently active property file (see SETPROP) for a certain command. If no command specific property file is activated, a property file is generated (see GENPROP), updated and activated (see SETPROP). :> FLCL CHGPROP command [path[=value]]*

CLEP Command line executor and parser

CLEP-ABOUT Built-in command to show information about the software product: version number, dependence of external libraries and more.

CLEP-COMMANDS The top level arguments of a program using CLEP to implement its command line interface. With CLEP the commands a program supports are defined within a tree of tables whoose values are automatically stored in a preinitialized data structure, which is mapped to the parameter structure of the subprogram, which provides the actual functionality.

CLEP-CONFIGURATION Built-in command to show the current configuration: owner, property file names, trace options and more.

CLEP-TRACE Build-in command to set the tracing options of CLEP: switch tracing ON or OFF and set a file name where trace output will be written to. When no file name is set, the trace is redirected to stdin/stdout.

COMMAND LINE ARGUMENTS Format and meaning of the parameters depend upon the command issued. Parameters may be either arguments or options. The program is typically invoked by entering its name followed by the command-line arguments.

COMMAND-LINE INTERFACE (CLI) A command-line interface (CLI), also known as command-line user interface, console user interface, and character user interface (CUI), is a means of interacting with a computer program where the user (or client) issues commands to the program in the form of successive lines of text (command lines).

COMMAND LINE INTERPRETER The program interpreting the commands typed by a user. On Unix this is called a shell (bash, csh, ksh, or zsh). On Windows® PowerShell, cmd.exe or COMMAND.COM.

COMMAND LINE PARAMETERS Command line parameters (also called flags) are parameters to a command entered behind the command name. They are separated from the command and further parameter by at least one space character. Parameters can be distinguished between options and arguments.

COMMAND LINE PARSER With a command line parser you easily define strongly typed command line arguments, allow automatic parsing of command line arguments and mapping of the values to the properties of your objects.

COMMAND LINE PARSER LIBRARY The Command Line Parser Library can offer to many console applications a clean and concise API for manipulating command line arguments and related tasks. You can use a help screen with a high degree of customization and report the syntax errors of the command to the user.

COMMAND-LINE REFERENCE The Command-Line Reference allows you to explore the documentation resources and other technical information that is necessary to learn about the command shell, and to automate command-line tasks by using scripts or scripting tools.

COMMAND PROMPT A command prompt (or just prompt) is a sequence of (one or more) characters used in a command-line interface to indicate readiness to accept commands.

CONFIGURATION FILE Configuration files contain the settings for a computer program. Usually this are plain text files. The structure of the files is application specific.

CONSOLE APPLICATION A console application is a computer program which is usually executed via a text-only computer interface (text terminal), the command line interface.

DEL Delete

DELETING OF PROPERTY FILE If you use DELPROP without further addition, you will be able to delete the general property file. Using this command with a specific command name you can delete the specific property file from the configuration data but only for the current owner. :> FLCL DELPROP :> FLCL DELPROP command

GEN Generate

GENERATION OF PROPERTY FILE With the help of command GENPROP you can generate a property file for the CLEP, which enables you to specify default properties of the processed command line. The actual owner is able to activate the property file. For each parameter a property entry is written to a text file. The definition of properties can be found directly at the top of the text file. Note that a special property file overrules the general property file and that you should delete expendable properties.

HELP Online Computer help links you in general to a database or similar organization of hundreds of different pages with technical information which shall help the user to run the program. The CLP help function makes extensive information about each parameter available for the user which is adjustable on the command line. By this means you get a structured list of arguments:

- keyword - the keyword for the argument
- (TYPE: . . .) - the type of this argument
- help message - the short help text
- (PROPERTY: [. . .]) - the current property value (optional) The keywords have to be unique for this list, but they can be still used in their short form. The writing can be case sensitive if wished requiring an additional line to mark the appropriate letter. The **path** is a dotted keyword list as per syntax of the command (command.para1.para2.para3). A detailed description of the manual pages can be printed with keyword **MAN**.

LEXEM Built-in function to show how a number, a string, a keyword and other values are defined in their correct command syntax. It uses regular expressions for the tokens of the command.

MAN Manual

MANPAGE A manual page is the local software documentation usually found on Unix operating systems. The CLP-manpage function shows the manual pages, all commands and built-in functions and if available for each argument up to selections/constant definitions. It writes the relating section from the user manual to the screen (doctype: **book**). If a filename is provided then the doctype **manpage** of ASCIIDOC is written to the file. Manpages can be created with the tool found under ASCIIDOC.

OBJECT An object argument has one or more members. Each of them can be an object or an overlay or a simple value. It is implemented by CLEP as a C struct.

OWNER The owner of a file in UNIX/LUNIX or WINDOWS controls how permissions are set on the file and to whom permissions are granted. If you generate a property file for the CLEP, you will be able to change the owner with the command SETOWNER. Thus you can run your own management of automated commands. This is a powerful concept when working in different or separated environments.

OVERLAY An overlay argument may only have one out of several values at a time. It is implemented by CLEP as a C union.

PATH Each argument of a CLEP program has a position in the command tree structure. This position is specified by the path: a DOT separated list of argument names, very similar to a file system path.

PROP Property

PROPERTY FILES Property files are used to configure applications. The project configuration data or settings are stored in a property file which is activated by the owner of the file.

SETTING OF PROPERTY FILE The file name is assigned to the actual global property file when you specify a command with no parameters. On the other hand the file name is assigned to its local property file when you define a command with appropriate options (SETPROP). The property file can be made with GENPROP and will be activated for the actual owner. If you have successfully created the property file you can activate it, but it will be still inactive until you alter the default settings.

SHELL SCRIPT Commands can be stored in a text file called shell script. The Shell Script is itself a command and can be called like a system command also with parameters.

SYNTAX Built-in function of CLEP to show the syntax of a program. Each built command string must follow special rules and refers to an argument list which is interpreted by a shell or command-line interpreter when running the program. Commands can be from type object or overlay. In dependence from command depth the syntax is printed in one line or in a structured form.

TRACING If switched ON CLEP tracing shows verbose information while the command line is evaluated.

UTF8 TO MSDOS COMMAND LINE ENCODING Windows cmd line by default does not support UTF8 but understands it. If you want to output multibyte characters in your cmd line, you will have to open a command prompt window, change the properties of the window to the Lucida Console True Type font. Then you must run "chcp 65001" on the command prompt to be able to output UTF8.

ADC Abbreviation for Advanced Data Compression, which is the default compression algorithm of FLAM4.

AES The Advanced Encryption Standard (AES) is a symmetric crypto system, the successor of DES and 3DES. It uses the Rijndael algorithm with a fixed block size of 128 bits and a variable key length of 128, 192 or 256 bits, providing a high grade of security.

ASCII The American Standard Code for Information Interchange (ASCII) is a 7-Bit encoding, which is the source for later character sets based on more bits.

BIG ENDIAN Endianess refers to the term byte order. A big-endian machine stores the most significant byte first, i.e. at the lowest byte address. This behavior is most important when two computers with different byte orders communicate with each other. The big-endian format is used by Motorola 6800 and 68k, Xilinx Microblaze, IBM POWER, System/360, System/370, ESA/390, z/Architecture and PDP-10.

BOM The Byte Order Mark (BOM) characterizes in which order the bytes are interpreted. This is important when exchanging data between different systems. For UTF-16 and UTF-32 a BOM sign at the beginning of the data stream is necessary, as more than one byte is used for the encoding of one character. UTF-8 uses the BOM sign "EF BB BF", which is used to mark a text file as UTF-8, even though byte order is not issue here.

BOMUTF With this option of the FLCL, you can determine the correct UTF CCSID from the byte order mark (BOM).

BOMUCS With this option of the FLCL, you can determine the correct UCS CCSID from the byte order mark (BOM). BYTE ORDER: The byte order is the order in which bytes are interpreted for entities that consist of multiple bytes, e.g. Unicode characters. This is important when exchanging data between different systems. A big-endian machine stores the most significant byte first, i.e. at the lowest byte address. A little-endian machine, however, stores the least significant byte first.

CANONICAL EQUIVALENCE Canonical equivalence is a fundamental equivalence between individual Unicode characters and sequences of Unicode characters.

CASEMETHOD FLCL offers a method for converting unicode characters to uppercase, lowercase or some special representation (CASE=UPPER/LOWER).

CCSID This is an abbreviation used by IBM for "Coded Character Set Identifier". The encoding of a specific code page is referred to by a 16-bit number.

CHAR Abbreviation in FLCL for the conversion of character sets.

CNV Abbreviation in FLCL for conversion.

CODEPAGE IBM introduced a numbering scheme to character encodings (16 bits). This scheme was taken over by other vendors like Microsoft, SAP or Oracle Corporation. The term Codepage is used synonymously for character encoding or character set.

CODE POINT A code point or code position is a single character in a list of characters represented as numerical values that create the code space. The Unicode code space has a total size of 17 * 65536 code points. The same code space may have several codepages.

COMPATIBILITY EQUIVALENCE One speaks of compatibility equivalence of two sets of characters if their full compatibility decompositions are identical.

CONV The sub-program FLUC offers all possible conversion and formatting methods via the CONV command. These are various read and write procedures, many useful conversion tools and some special data formatting features.

CX7 Compression encoding as printable characters in FLAM4

CX8 Compression encoding as binary data in FLAM4

DIACRITIC A diacritic is also called diacritical mark, diacritical point or diacritical sign. Typical diacritical marks are the acute (´) and grave (`), which are often called accents. Diacritical marks may be a dash, dot or tick above or below a letter, or in some other position such as within the letter or between two letters.

EBCDIC The Extended Binary Coded Decimals Interchange Code (EBCDIC) was developed by IBM as 8-Bit-encoding, which, however, does not use all code words. EBCDIC originates from the older 4-Bit-Code BCD (Binary Coded Decimals) and is applied solely on Mainframes. The EBCDIC codepages are available in German, US, Swedish, Arabian, Latin1 and other languages. Moreover, there exists a Unicode set of characters based on EBCDIC and it is called UTF-EBCDIC.

FIO The C programming language has a general mechanism for reading and writing files abstracting all Input/Output to streams of bytes. The operating system offers redirection which can be very useful. Examples are the standard input stream, standard output stream and standard error stream.

FLAM Frankenstein Limes Access Method. FLAM is our access method for encrypted and compressed data. In particular, FLAM is a utility for converting, compressing, encrypting (with AES (Advanced Encryption Standard)), signing and formatting of mass data with simple interfaces for integration in all kinds of solutions. All this is done with a minimum of CPU consumption, utilization of co-processors and the use of professional key management solutions. Ordinarily, FLAM provides - when used as sub-program - the commands COMP and DECO and makes use of an element interface for the subsystems/plugins/drivers and also the sub-program.

FLAM4 This component writes a Flamfile record-oriented. You can state the name of a Flamfile or that of a member.

FLAMFILE The FLAMFILE is composed like a package file, as multiple files can be compressed. It is structured clearly and divided hierarchically in records, segments and members. In this connection a segment is referred to the single matrix of a compressed file and a member to the entity of segments that are part of the compressed file. In order to ensure the universal exchangeability, the FLAMFILE has to be available in a format that is representable on all operating systems (so-called heterogeneous compatibility). Each compressed data segment is unique referring to data content, environment and time. This allows - despite compression/encryption - an access down to the record-level, without decompressing/decrypting.

FLCL Frankenstein Limes Command Line. FLCL offers a command line for the three sub-programs FLAM, FLUC and FLIES. It is a quasi-batch utility for the sub-programs. Moreover, FLCL is a universal data format and character set converter, which can be included easily in professional internationalization projects. It supports different input and output formats (GZIP, PDF, B64...) and can be integrated into scripts or as sub-program. More than 150 codepages are supported. FLCL can be applied to various commonly used platforms if working with processable data streams or file I/O.

FLIES Frankenstein Limes Integrated Extended Security. The sub-program FLIES governs the compressed and encrypted segments using commands like FIND and CHNGE. In particular, it is useful for processing Flamfiles, which however cannot be decrypted or decompressed. For instance the data in the file header is managed by FLIES to re-encrypt the file, change the data access, use as backup, archive or as mediator/headend.

FLUC Frankenstein Limes Universal Converter. FLUC is a universal text converter with special focus on the high requirements in professional internationalization projects, where Mainframes and external applications are used as central backend.

FROM-CODE The FROM-Code specifies the character encoding of the input data that used as source encoding for character conversion.

GZIP GZIP is a compression program for nearly every operation system and uses the DEFLATE algorithm from the ZLIB library. GZIP is the short form for GNU zip. The used algorithm is a combination of LZ77 and Huffmann encoding. The source code is licensed under the GPL. The program has 9 compression steps providing either optimal run times or compression ratios. The compressed file has the ending ".gz". As GZIP compresses only single files. Several files must first be combined with (for example) "tar" to a tarball which can then be compressed to a ".tar.gz" file. GZIP files have a 10-byte header, containing a magic number, a version number and a timestamp. With FLCL you are able to read and write GZIP files, to mark content as text data in the header, set operating/file system or modified time (seconds since 1970).

HOST see MAINFRAME

ICNV Abbreviation in FLCL for iconv conversion with simplified commands on the command line.

ICONV With the standard program iconv, characters can be translated between different encodings on UNIX-like systems. As iconv is now part of the GNU C Library, actual Linux systems are equipped with iconv. Under Windows®, iconv is provided only as DLL or static library, which is under the license of the GPL and can be used via Cygwin or GnuWin32. Iconv is also supported as Dynamic Link Library for PHP.

ISO-8859 The ISO-8859 norm family contains some useful 8-bit character sets (e.g. Latin-1 to Latin-15). The frequently used Latin-1 character set is identical with the first 256 characters of the Unicode character set. These codepages are also called extended Ascii.

LITTLE ENDIAN A little-endian machine stores the least significant byte first. The endianness of a 16, 32 or 64 bits word or even bits can occur in some Unicode representations. Also mixed forms of endianess are possible, for example when the byte order within a 16 bits word differs from that of a 16 bits word within a 32 bits word. The little-endian format is used by x86 (including x86-64), 6502 (including 65802, 65C816), Z80 (including Z180, eZ80 etc.), MCS-48, 8051, DEC Alpha, Altera Nios II, Atmel AVR, SuperH, VAX, and PDP-11.

LOGGING In FLCL various messages (warnings, errors, debug information, important information, input data) can be logged. The messages can be redirected from stderr to stdout or a file.

MAINFRAME A mainframe is a very complex and powerful computer system. Its capacity is far beyond that of a common Personal Computer and that of most servers. They are applied for the highly reliable processing of mass data, mass transactions and company critical data and gain importance again in connection with server consolidation and cloud computing.

MODE With the keyword MODE in FLCL you can select the mode of operation.

NDC Abbreviation for No Data Compression, which can be considered as copying data in FLAM4.

NFC This is Normalization Form C (Canonical Decomposition, followed by Canonical Composition).

NFD This is Normalization Form D (Canonical Decomposition).

NFKC This is Normalization Form KC (Compatibility Decomposition, followed by Canonical Composition).

NFKD This is Normalization Form KD (Compatibility Decomposition).

Normalization The transformation of a character sequence in one of the four normal forms is called normalization. Normalization is necessary, because there is more than one differing possibility for many characters to be represented as a sequence of Unicode characters. The four normal forms consist of one form for the canonical equivalence, one for the compatible equivalence, and both can be in a "composed" or "decomposed" form.

OS Abbreviation for operation/file system in the header

PADDING Padding are bytes inserted to align the data to certain lengths or storage positions. Padding bytes are most commonly appended after the the last data byte. When reading padded data, the padding bytes are discarded.

PLATFORM NEUTRAL The components of FLAM are platform neutral. This means that they work independently from the used operating system. Operating systems can be for example LINUX, UNIX-like systems, WINDOWS® or z/OS®.

SEPA The Single Euro Payments Area (SEPA) is a project to harmonize the way retail payments are made and processed in Euro. The motivation behind it is to make payments in Euro and across the EU as fast, safe and efficient as national payments are now. SEPA shall enable customers to make cashless Euro payments to anyone in the EU, for instance by credit transfer, direct debit or debit card.

SUB CHARSET A Sub Charset is a partial quantity of a character set. For instance the new identity card issued in Germany (nPa) requires that each printed character - even the diacritical characters - is "composed". It means that they remain in their original form and are not "decomposed" into several bytes. For that reason, the Sub Charset "Latin in Unicodes" with some more than 500 letters is used.

SUBCHR FLCL can substitute characters that are unsupported in their target encoding by system specific characters (e.g. 0x1A for ASCII or 0x3F for EBCDIC pages) or by arbitrary characters. In the first case, you must use SUBCHR=SYSTEM, in the other you can define your own substitution characters. On z/OS®, the utility iconv is available on the system and replaces all invalid characters by system specific characters, including the BOM sign. Note that it does not stop in contrast to LINUX or WINDOWS® but completes conversion.

SUBSTITUTE If a character exists in the source code but not in the target code, it will be converted to a converter-defined substitute character.

SURROGATE The Unicode code space is divided into 17 planes of 65536 code points each. Some code points have no character value, some are reserved for private use, and some are permanently reserved as non-characters. UTF-16 with code points beyond these planes is encoded in 16 bit surrogate pairs (High surrogate and low surrogate), which are in fact replacement characters. UTF-16 surrogate values are illegal in UTF-32. Note that the conversion of UTF-16 strings into UTF-8 byte sequences requires that the High and Low surrogate have to be combined correctly before they are transfered into UTF-8.

SYSTEM TABLE FLCL makes use of a pre-loaded system substitution table for the transliteration of characters. This is comparable to "translit.h" of iconv under LINUX.

TO-CODE With few utilities you can convert text from one encoding to another. The TO-Code specifies the output encoding that characters are converted to.

TRANSCRIPTION To transcribe the expression of a language means by dictionary, to represent it in another writing system rather in that, in which the language is usually written respectively in which it is available in written form. For the term of transcription it does not matter, in which medium the expression is available. Note that the transliteration of an expression is a particular type of transcription.

TRANSLATION This is the transfer of a (mostly written) and fixed text from the original language into any target language; the process will be also denoted as "Translating".

TRANSLITERATION To transliterate an expression represented in writing S1 into a writing system S2 means, that the characters contained in the expression are exchanged from S1 to S2 according to a fixed rule. In the simplest case, this can be a table which assigns each character of S1 a character of S2. Frequently, there are complications for the simple case, if for example a single character is assigned to a digraph or vice versa or if a character is mapped in context A onto a specific character, in context B however onto another character. For example: in a payment order from Germany, the name of the receiver must be written explicitly in Latin letters so that a transliteration of the original text with help of a table is required.

UNICODE The so-called Unicode is an attempt, started in the 90s, to standardize a single character set that eliminates the need for different and incompatible encodings in different countries and cultures. The text should be always readable with a Unicode-supporting appropriate editor. This allows an error free data exchange (for the purpose of readability) also on a global and international level. For this character set, 8 bits (1 byte) are no longer sufficient to for a representation of every possible character. The length of characters encoded in Unicode can be between 1 and 4 bytes. ISO 10646 is a practically equal notation of the Unicode character set called Universal Character Set (UCS).

USER TABLE For character set conversion in FLCL you can define your own user substitution table to transliterate those characters that are not handled respectively replaced by the system table.

UTF-8 The 8-Bit Unicode Transformation Format is the most commonly used encoding for Unicode characters. The term Unicode is used equivalent to UCS. UTF-8 has a central meaning in the global character conversion for internet protocols.

UTF-16 In UTF-16 an encoded byte sequence with a length of 2 bytes is assigned to each Unicode character. UTF-16 is used in Java for internal character sequence representation. UTF-16 is the eldest format of the Unicodes.

UTF-32 UTF-32 (UCS-4) is encoded with 4 bytes and therefore the most simple Unicode representation, as other formats need variable byte lengths for their representation. One disadvantage of UTF-32 is the greater space requirement compared to UTF-8 and ISO-8859 character sets and the missing backward compatibility to ASCII.

VR8 Enhanced compression encoding as binary data in FLAM4

Appendix A

LEXEM

The regular expressions used by the command line compiler are shown below. This content is printed on the screen with the built-in function *LEXEM*.

To describe the regular expression the following character classes are used:

- :print: - all printable characters
- :space: - all blanks, tabs and new lines
- :cntr: - all control characters
- :alpha: - all letters ()
- :alnum: - all letters and digits [0-9]
- :digit: - all decimal numbers [0-9]
- :xdigit: - all hexadecimal numbers

All comments and separators are ignored by the scanner and not passed with the token to the parser. This means the separators (SEP) are used in the grammar but will never be part of or own tokens.

If a binary entry for a string is possible and no classification is done it is taken as a binary string in local character set representation (c) without null termination. This default can be changed. In this case the help message for this argument should be described the default interpretation of the binary string. For binary strings it is still possible to enter a null-terminated string, but then the prefix **s** must be used. If no binary entry is feasible, only a null-terminated string in local character representation (s) can be built and taken per default.

```
Lexemes (regular expressions) for argument list or parameter file
--| COMMENT   '#' [:print:]* '#'                                (will be ignored)
--| LCOMMENT  ';' [:print:]* 'nl'                               (will be ignored)
--| SEPARATOR [:space: | :cntr: | ',']*                  (abbreviated with SEP)
--| OPERATOR  '=' | '.' | '(' | ')' | '[' | ']'  (SGN, DOT, RBO, RBC, SBO, SBC)
--| KEYWORD   ['-'['-']][:alpha:]+[:alnum: | '_' | '-']*    (always predefined)
--| NUMBER    ([+|-]  [ :digit:]+)  |                          (decimal (default))
--| num       ([+|-]0b[ :digit:]+)  |                                    (binary)
--| num       ([+|-]0o[ :digit:]+)  |                                     (octal)
--| num       ([+|-]0d[ :digit:]+)  |                                   (decimal)
--| num       ([+|-]0x[ :xdigit:]+) |                                (hexadecimal)
--| num       ([+|-]0t(yyyy/mm/tt.hh:mm:ss)) |  (relativ (+|-) or absolut time)
--| FLOAT     ([+|-]  [ :digit:]+.[:digit:]+e|E[:digit:]+) | (decimal(default))
--| flt       ([+|-]0d[ :digit:]+.[:digit:]+e|E[:digit:]+)            (decimal)
--| STRING         ''' [:print:]* ''' |          (default (if binary c else s))
--| str       [s|S]''' [:print:]* ''' |                (null-terminated string)
--| str       [c|C]''' [:print:]* ''' |  (binary string in local character set)
```

```
--| str       [a|A]''' [:print:]* ''' |                (binary string in ASCII)
--| str       [e|E]''' [:print:]* ''' |               (binary string in EBCDIC)
--| str       [x|X]''' [:print:]* ''' |         (binary string in hex notation)
--|           Strings can contain two '' to represent one '
--|           Strings can also be enclosed in " instead of '
--|           Strings can directly start behind a '=' without enclosing '/"
--|              In this case the string ends at the next separator or operator
--|              and keywords are preferred. To use keywords, separators or
--|              operators in strings, enclosing quotes are required.
--| SUPPLEMENT     '"' [:print:]* '"' |   (null-terminated string (properties))
--|           Supplements can contain two "" to represent one "
--|           Supplements can also be enclosed in ' instead of "
```

Appendix B

GRAMMAR

Below you can see the grammar used by the command line compiler. This content is printed on the screen if the built-in function *GRAMMAR* is used.

There are two context free grammars, one is used for the command line the other for the parameter file. The parameter file contains mainly the same argument list as the command line. The property file defines the default settings. For this a different grammar is applied.

Properties can be set for each argument, i.e. for arguments not used or arguments in overlays. Based on that, properties are set by using the path (dotted keyword list).

A property is always a *SUPPLEMENT* for the argument list on the command line or parameter file. Thus all defined properties have the same effect as arguments on the command line. A property sets this argument to the corresponding value if it is not defined on the command line.

```
Grammar for argument list, parameter file or property file
--| Command Line Parser
--| command        -> ['('] parameter_list [')']       (main=object)
--|                |  ['.'] parameter                   (main=overlay)
--| parameter_list -> parameter SEP parameter_list
--|                |  EMPTY
--| parameter      -> switch | assignment | object | overlay | array
--| switch         -> KEYWORD
--| assignment     -> KEYWORD '=' value
--| object         -> KEYWORD ['('] parameter_list [')']
--|                |  KEYWORD '=' STRING # parameter file #
--| overlay        -> KEYWORD ['.'] parameter
--|                |  KEYWORD '=' STRING # parameter file #
--| array          -> KEYWORD '[' value_list   ']'
--|                |  KEYWORD '[' object_list  ']'
--|                |  KEYWORD '[' overlay_list ']'
--| value_list     -> value SEP value_list
--|                |  EMPTY
--| object_list    -> object SEP object_list
--|                |  EMPTY
--| overlay_list   -> overlay SEP overlay_list
--|                |  EMPTY
--| value          -> NUMBER | FLOAT | STRING | KEYWORD
--| A list of objects requires parenthesis to enclose the arguments
--|
--| Property File Parser
--| properties     -> property_list
--| property_list  -> property SEP property_list
--|                |  EMPTY
--| property       -> keyword_list '=' SUPPLEMENT
--| keyword_list   -> KEYWORD '.' keyword_list
```

```
--|                 |  KEYWORD
--| SUPPLEMENT is a string in double quotation marks ("property")
```

Appendix C

PROPERTIES

Below you can see the properties. This content can be written to the global property file with the built-in function *GENPROP*.

The first comment shows the valid root path for the property file. A property file can contain properties for more than one root path. The property file parser will only interpret properties where the root matches, i.e. if the property file is assigned to a certain owner and possibly to a certain command, then only properties for this owner, the program itself and possibly the command will be accepted, all other properties are ignored.

The second long comment gives some help for handling a property file. Thereafter the generated properties for this root are listed for all supported commands. The format is:

```
path = SUPPLEMENT # Type: tttttt Help: hhh...hhh #
```

The path denotes the unique dotted keyword list for one argument. The *SUPPLEMENT* is a string normally in double quotation marks containing the value list for this argument in accordance with the syntax of the command line parser (strings then normally in single quotation marks, a opposite usage of the quotation marks is possible but not recommended). This means for example that a string that requires apostrophes must be included in apostrophes (`"'...'"`). The supplement string is returned by the corresponding command line parser if no value for this argument is entered on the command line or is included in the parameter file.

Behind the property definition the type and help message are printed as comment to help in editing the generated property file. The property values can be related to the current default settings at time of generation, what means that if hard coded default values or another property file is already in use, then these values are taken over into the new property file.

The property file can contain parameter which are not available on the command line. Such parameter are only documented in the property file. On the other side not all command line parameter are available as property.

If a description for a property argument is available which is not a part of the user manual this description is written as comment in front of the corresponding property definition.

```
# Property file for: LIM.flcl #

#------------------------------------------------------------------ #
#                                                                   #
# The property file can be used to overwrite the default values     #
# used if the argument is not defined on the command line.          #
#                                                                   #
#------------------------------------------------------------------ #
#                                                                   #
# The values is assigned between "" conforming to the syntax.       #
#   Example NUMBER array  : "4711, 1, 2, 3"                         #
#   Example STRING array  : "'str1' x'303000''str2',a'abc'"         #
#   Example FLOAT array   : "123.34, 1.58, PI" (PI as constant)     #
#                                                                   #
# The last example shows that one can use key words if a list       #
# of constants is defined for this argument.                        #
```

```
#                                                                     #
# For a SWITCH one can use the special key words ON/OFF to enable or #
# disable the SWITCH or define the value as a number                  #
#   Examples for a SWITCH : "ON"/"OFF"/"4711"                         #
#                                                                     #
# For an OBJECT only the special key word INIT can be used to ensure #
# that this OBJECT will be initialized if it was not set on the       #
# command line. If INIT is not set, then the elements of the OBJECT   #
# will be initialized only if you type at least obj() in the command #
# line. If you want an object to be initialized when you don't use it#
# on the command line then the special key word must be set.          #
#   Examples for an OBJECT: ""/"INIT"                                 #
#                                                                     #
# For an OVERLAY it is the same logic as for objects but the keyword #
# of the element to initialize must be used if the overlay is not     #
# set in the command line.                                            #
#   Example OVERLAY array : "LINE TRIANGLE LINE RECTANGLE"            #
#   Example for a OVERLAY : "RECTANGLE"                               #
#                                                                     #
#-------------------------------------------------------------------- #


# DESCRIPTION for LIM.flcl.INFO.NET:
With the overlay NET you can choose the protocol for the remote read
access. Below You can find the currently supported protocols and the
corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.INFO.NET="" # TYPE: OVERLAY HELP: Network / Remote access #

# DESCRIPTION for LIM.flcl.INFO.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.INFO.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.INFO.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.INFO.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.INFO.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.INFO.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #

# DESCRIPTION for LIM.flcl.INFO.NET.IPS:
```

```
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.INFO.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.INFO.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.INFO.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.INFO.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.INFO.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #

# DESCRIPTION for LIM.flcl.INFO.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.INFO.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.INFO.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
 LIM.flcl.INFO.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.INFO.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.INFO.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #

# DESCRIPTION for LIM.flcl.INFO.GET:
With the overlay GET you can choose which information will be printed.


 #
 LIM.flcl.INFO.GET="" # TYPE: OVERLAY HELP: Specify what you want to get #

 LIM.flcl.INFO.GET.CCSIDS="" # TYPE: SWITCH HELP: List of supported CCSIDs #
 LIM.flcl.INFO.GET.ENCODINGS="" # TYPE: SWITCH HELP: List of all encodings #
 LIM.flcl.INFO.GET.FKME="" # TYPE: OBJECT HELP: List informations, help and documentation  ↩
    of FKME #
 LIM.flcl.INFO.GET.FKME.LIBRARY="" # TYPE: STRING HELP: Library name [libfkme] #
 LIM.flcl.INFO.GET.FKME.FUNCTION="" # TYPE: STRING HELP: Function name (entry) #
 LIM.flcl.INFO.GET.FKME.HELP="" # TYPE: SWITCH HELP: Get help message #
```

```
LIM.flcl.INFO.GET.FKME.DOCU="" # TYPE: SWITCH HELP: Get documentation #
LIM.flcl.INFO.GET.FILE="" # TYPE: STRING HELP: Information about a file (flam4/gzip/xz/ ↩
    bzip2/...) #

LIM.flcl.INFO.FMT="" # TYPE: NUMBER HELP: Specify formatting of informations [LIST] #
LIM.flcl.INFO.OUT="" # TYPE: STRING HELP: Name of the output file [LOG] #
LIM.flcl.INFO.DIR="" # TYPE: OBJECT HELP: Defines how to handle directories and various  ↩
    file types if wildcards used #
LIM.flcl.INFO.DIR.LINK="" # TYPE: SWITCH HELP: Include symbolic links #
LIM.flcl.INFO.DIR.ALIAS="" # TYPE: SWITCH HELP: Resolve data set aliases #
LIM.flcl.INFO.DIR.HIDDEN="" # TYPE: SWITCH HELP: Include hidden files #
LIM.flcl.INFO.DIR.RECURSIVE="" # TYPE: SWITCH HELP: Include sub directories #
LIM.flcl.INFO.DIR.ARCHIVE="" # TYPE: SWITCH HELP: Include archives (FLAMFILEs, ZIP, ...) #
LIM.flcl.INFO.DIR.ONEFILESYSTEM="" # TYPE: SWITCH HELP: Don't cross file system boundaries ↩
     #
LIM.flcl.INFO.DIR.TAPE="" # TYPE: SWITCH HELP: Include files deposited on tapes #
LIM.flcl.INFO.LOG="" # TYPE: OBJECT HELP: Logging Parameter #
LIM.flcl.INFO.LOG.MESSAGE="" # TYPE: OBJECT HELP: Select type of messages to log #
LIM.flcl.INFO.LOG.MESSAGE.NONE="" # TYPE: SWITCH HELP: No messages will be logged #
LIM.flcl.INFO.LOG.MESSAGE.ERROR="" # TYPE: SWITCH HELP: Error messages will be logged #
LIM.flcl.INFO.LOG.MESSAGE.ERRTRACE="" # TYPE: SWITCH HELP: Error trace will be logged #
LIM.flcl.INFO.LOG.MESSAGE.WARNING="" # TYPE: SWITCH HELP: Warnings will be logged #
LIM.flcl.INFO.LOG.MESSAGE.NOTICE="" # TYPE: SWITCH HELP: Notices will be logged #
LIM.flcl.INFO.LOG.MESSAGE.DEBUG="" # TYPE: SWITCH HELP: Debug information will be logged #
LIM.flcl.INFO.LOG.MESSAGE.ABOUT="" # TYPE: SWITCH HELP: About information will be logged #
LIM.flcl.INFO.LOG.MESSAGE.VERSION="" # TYPE: SWITCH HELP: Version information will be  ↩
    logged #
LIM.flcl.INFO.LOG.MESSAGE.INPUT="" # TYPE: SWITCH HELP: Input data will be logged ( ↩
    dangerous for passwords) #
LIM.flcl.INFO.LOG.MESSAGE.PARAMETER="" # TYPE: SWITCH HELP: Parsed parameters will be  ↩
    logged #
LIM.flcl.INFO.LOG.MESSAGE.STATISTIC="" # TYPE: SWITCH HELP: Statistics will be logged #
LIM.flcl.INFO.LOG.MESSAGE.INFO="" # TYPE: SWITCH HELP: Requested information will be  ↩
    logged #
LIM.flcl.INFO.LOG.MESSAGE.LICID="" # TYPE: SWITCH HELP: License ID will be logged #
LIM.flcl.INFO.LOG.MESSAGE.LICENSE="" # TYPE: SWITCH HELP: License summary will be logged #
LIM.flcl.INFO.LOG.MESSAGE.PROGRESS="" # TYPE: SWITCH HELP: Progress bar will be logged #
LIM.flcl.INFO.LOG.MESSAGE.ALL="" # TYPE: SWITCH HELP: All messages are logged #
LIM.flcl.INFO.LOG.MESSAGE.DEFAULT="" # TYPE: SWITCH HELP: Relevant messages are logged #
LIM.flcl.INFO.LOG.MESSAGE.MINIMAL="" # TYPE: SWITCH HELP: Only important messages are  ↩
    logged #
LIM.flcl.INFO.LOG.MESSAGE.ERRONLY="" # TYPE: SWITCH HELP: Only errors and error trace are  ↩
    logged #
LIM.flcl.INFO.LOG.DESTINATION="" # TYPE: OVERLAY HELP: Destination for log messages #
LIM.flcl.INFO.LOG.DESTINATION.STREAM="" # TYPE: OBJECT HELP: Log to a stream #
LIM.flcl.INFO.LOG.DESTINATION.STREAM.FORMAT="" # TYPE: NUMBER HELP: Format of log message  ↩
    [STANDARD] #
LIM.flcl.INFO.LOG.DESTINATION.STREAM.IDENT="" # TYPE: STRING HELP: Identifier used to  ↩
    identify log entries #
LIM.flcl.INFO.LOG.DESTINATION.STREAM.STDOUT="" # TYPE: SWITCH HELP: Log messages to STDOUT ↩
     instead of STDERR #
LIM.flcl.INFO.LOG.DESTINATION.FILE="" # TYPE: OBJECT HELP: Log to a flat file #
LIM.flcl.INFO.LOG.DESTINATION.FILE.FORMAT="" # TYPE: NUMBER HELP: Format of log message [ ↩
    STANDARD] #
LIM.flcl.INFO.LOG.DESTINATION.FILE.IDENT="" # TYPE: STRING HELP: Identifier used to  ↩
    identify log entries #
LIM.flcl.INFO.LOG.DESTINATION.FILE.NAME="" # TYPE: STRING HELP: Filename for appending log ↩
     messages #
LIM.flcl.INFO.LOG.DESTINATION.SYSTEM="" # TYPE: OBJECT HELP: Use system logging facility #
LIM.flcl.INFO.LOG.DESTINATION.SYSTEM.IDENT="" # TYPE: STRING HELP: Identifier used to  ↩
    identify log entries #
LIM.flcl.CONV.READ="" # TYPE: OVERLAY HELP: Inbound parameter to read data #
```

```
 LIM.flcl.CONV.READ.BINARY="" # TYPE: OBJECT HELP: Read binary data from a file #

# DESCRIPTION for LIM.flcl.CONV.READ.BINARY.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.CONV.READ.BINARY.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read  ↩
     access #

# DESCRIPTION for LIM.flcl.CONV.READ.BINARY.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.BINARY.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.CONV.READ.BINARY.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.BINARY.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.READ.BINARY.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.BINARY.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.READ.BINARY.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.
```

```
 #
 LIM.flcl.CONV.READ.BINARY.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.CONV.READ.BINARY.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.BINARY.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.READ.BINARY.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.BINARY.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.READ.BINARY.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.BINARY.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.CONV.READ.BINARY.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] ↩
      #
 LIM.flcl.CONV.READ.BINARY.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.CONV.READ.BINARY.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.BINARY.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.CONV.READ.BINARY.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''==stdin ↩
     ] #
 LIM.flcl.CONV.READ.BINARY.BLKSIZE="" # TYPE: NUMBER HELP: Block size to read [AUTO] #
 LIM.flcl.CONV.READ.BINARY.DECODE="" # TYPE: NUMBER HELP: Decode encoded, encrypted or  ↩
     compressed data [0=NONE] #
 LIM.flcl.CONV.READ.BINARY.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
 LIM.flcl.CONV.READ.BINARY.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password  ↩
     decryption #
 LIM.flcl.CONV.READ.BINARY.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to  ↩
     encrypt the FLAMFILE ['********'] #
 LIM.flcl.CONV.READ.BINARY.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management  ↩
     extension (FKME) #
 LIM.flcl.CONV.READ.BINARY.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for  ↩
     FKME ['libfkme'] #
 LIM.flcl.CONV.READ.BINARY.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of  ↩
     FKME ['FKMESTD0'] #
 LIM.flcl.CONV.READ.BINARY.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for  ↩
     the call of FKME [''] #
 LIM.flcl.CONV.READ.BINARY.FDECODE="" # TYPE: OVERLAY HELP: Additional enforced decoding [ ↩
     NONE] #
 LIM.flcl.CONV.READ.BINARY.FDECODE.BASE64="" # TYPE: SWITCH HELP: Base64 decoding (RFC4648) ↩
      #
 LIM.flcl.CONV.READ.BINARY.FDECODE.BASE32="" # TYPE: SWITCH HELP: Base32 decoding (RFC4648) ↩
      #
 LIM.flcl.CONV.READ.BINARY.FDECODE.BASE16="" # TYPE: SWITCH HELP: Base16 decoding (RFC4648) ↩
      #
 LIM.flcl.CONV.READ.BINARY.RLDECO="" # TYPE: SWITCH HELP: Activate an additional record  ↩
     length decoding [FALSE] #
 LIM.flcl.CONV.READ.BINARY.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character  ↩
```

```
    handling [DETACH] #
 LIM.flcl.CONV.READ.BINARY.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO  ↩
    subsystem (only for z/OS) #
 LIM.flcl.CONV.READ.BINARY.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the  ↩
    subsystem #
 LIM.flcl.CONV.READ.BINARY.SUBSYSTEM.PARM="'" # TYPE: STRING HELP: Parameter list for the  ↩
    subsystem (comma separated) #
 LIM.flcl.CONV.READ.BINARY.FRCBLK="" # TYPE: SWITCH HELP: Enforce block orientation on  ↩
    record oriented devices [FALSE] #
 LIM.flcl.CONV.READ.BINARY.REMOVE="" # TYPE: SWITCH HELP: Remove original file after  ↩
    successful processing [FALSE] #
 LIM.flcl.CONV.READ.BINARY.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
    FL_PLATFORM for read operation #
 LIM.flcl.CONV.READ.BINARY.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash ↩
     values of input data #
 LIM.flcl.CONV.READ.BINARY.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.CONV.READ.BINARY.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
 LIM.flcl.CONV.READ.BINARY.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string  ↩
    #
 LIM.flcl.CONV.READ.BINARY.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
 LIM.flcl.CONV.READ.BINARY.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] ↩
     #
 LIM.flcl.CONV.READ.BINARY.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG ↩
    ] #
 LIM.flcl.CONV.READ.BINARY.CHECK="" # TYPE: SWITCH HELP: Activate additional optional  ↩
    verifications [FALSE] #
 LIM.flcl.CONV.READ.CHARACTER="" # TYPE: OBJECT HELP: Read character data from a file #

# DESCRIPTION for LIM.flcl.CONV.READ.CHARACTER.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.CONV.READ.CHARACTER.NET="" # TYPE: OVERLAY HELP: Network parameter for remote  ↩
    read access #

# DESCRIPTION for LIM.flcl.CONV.READ.CHARACTER.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.
```

```
 #
 LIM.flcl.CONV.READ.CHARACTER.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.CONV.READ.CHARACTER.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.CHARACTER.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.READ.CHARACTER.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.CHARACTER.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.READ.CHARACTER.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.CHARACTER.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.CONV.READ.CHARACTER.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.CHARACTER.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.READ.CHARACTER.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.CHARACTER.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.READ.CHARACTER.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.CHARACTER.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.CONV.READ.CHARACTER.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name  ↩
     [""] #
 LIM.flcl.CONV.READ.CHARACTER.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.CONV.READ.CHARACTER.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.CHARACTER.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.CONV.READ.CHARACTER.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''== ↩
```

```
    stdin] #
 LIM.flcl.CONV.READ.CHARACTER.BLKSIZE="" # TYPE: NUMBER HELP: Block size for read ↩
    operations [AUTO] #
 LIM.flcl.CONV.READ.CHARACTER.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID [ ↩
    auto detection] #
 LIM.flcl.CONV.READ.CHARACTER.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get ↩
    wrong BOM in the middle of data #
 LIM.flcl.CONV.READ.CHARACTER.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15) ↩
     by line feed (0x25) on input #
 LIM.flcl.CONV.READ.CHARACTER.DECODE="ALWAYS" # TYPE: NUMBER HELP: Decode encoded data ↩
    first (remove base encodings) #
 LIM.flcl.CONV.READ.CHARACTER.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] ↩
    #
 LIM.flcl.CONV.READ.CHARACTER.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password ↩
    decryption #
 LIM.flcl.CONV.READ.CHARACTER.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to ↩
    encrypt the FLAMFILE ['********'] #
 LIM.flcl.CONV.READ.CHARACTER.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management ↩
     extension (FKME) #
 LIM.flcl.CONV.READ.CHARACTER.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name ↩
    for FKME ['libfkme'] #
 LIM.flcl.CONV.READ.CHARACTER.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name ↩
    of FKME ['FKMESTD0'] #
 LIM.flcl.CONV.READ.CHARACTER.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for ↩
     the call of FKME [''] #
 LIM.flcl.CONV.READ.CHARACTER.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character ↩
    handling [DETACH] #
 LIM.flcl.CONV.READ.CHARACTER.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO ↩
    subsystem (only for z/OS) #
 LIM.flcl.CONV.READ.CHARACTER.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the ↩
    subsystem #
 LIM.flcl.CONV.READ.CHARACTER.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list for ↩
    the subsystem (comma separated) #
 LIM.flcl.CONV.READ.CHARACTER.FRCBLK="" # TYPE: SWITCH HELP: Enforce block orientation on ↩
    record oriented devices [FALSE] #
 LIM.flcl.CONV.READ.CHARACTER.REMOVE="" # TYPE: SWITCH HELP: Remove original file after ↩
    successful processing [FALSE] #
 LIM.flcl.CONV.READ.CHARACTER.LANG="" # TYPE: STRING HELP: Redefine environment variable ↩
    LANG for read operation #
 LIM.flcl.CONV.READ.CHARACTER.PLATFORM="" # TYPE: STRING HELP: Redefine environment ↩
    variable FL_PLATFORM for read operation #
 LIM.flcl.CONV.READ.CHARACTER.HASH="" # TYPE: OBJECT HELP: Generation or check of one way ↩
    hash values of input data #
 LIM.flcl.CONV.READ.CHARACTER.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.CONV.READ.CHARACTER.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum ↩
    verification #
 LIM.flcl.CONV.READ.CHARACTER.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex- ↩
    string #
 LIM.flcl.CONV.READ.CHARACTER.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains ↩
    Checksum as hex-string #
 LIM.flcl.CONV.READ.CHARACTER.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [ ↩
    GNU] #
 LIM.flcl.CONV.READ.CHARACTER.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [ ↩
    LOG] #
 LIM.flcl.CONV.READ.CHARACTER.CHECK="" # TYPE: SWITCH HELP: Activate additional optional ↩
    verifications [FALSE] #
 LIM.flcl.CONV.READ.TEXT="" # TYPE: OBJECT HELP: Read text data from a file #

# DESCRIPTION for LIM.flcl.CONV.READ.TEXT.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.
```

```
This feature requires a FLIES server.


 #
 LIM.flcl.CONV.READ.TEXT.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read  ↩
     access #

# DESCRIPTION for LIM.flcl.CONV.READ.TEXT.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.TEXT.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.CONV.READ.TEXT.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.TEXT.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.READ.TEXT.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.TEXT.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

# DESCRIPTION for LIM.flcl.CONV.READ.TEXT.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.TEXT.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.CONV.READ.TEXT.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.TEXT.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
```

```
 LIM.flcl.CONV.READ.TEXT.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.TEXT.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

# DESCRIPTION for LIM.flcl.CONV.READ.TEXT.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.TEXT.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.CONV.READ.TEXT.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
 LIM.flcl.CONV.READ.TEXT.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.CONV.READ.TEXT.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.TEXT.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

 LIM.flcl.CONV.READ.TEXT.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''==stdin]  ↩
     #
 LIM.flcl.CONV.READ.TEXT.BLKSIZE="" # TYPE: NUMBER HELP: Block size for read operations [ ↩
     AUTO] #
 LIM.flcl.CONV.READ.TEXT.RECLENGTH="" # TYPE: NUMBER HELP: Maximum record length (Host) /  ↩
     line length (Unix/Win) for text parsing [512] #
 LIM.flcl.CONV.READ.TEXT.SUPTWS="" # TYPE: SWITCH HELP: Suppress trailing whitespaces [ ↩
     FALSE] #
 LIM.flcl.CONV.READ.TEXT.NELDLM="" # TYPE: SWITCH HELP: Activate NEL (0x85) as delimiter  ↩
     for ASCII character sets [FALSE] #
 LIM.flcl.CONV.READ.TEXT.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID [auto  ↩
     detection] #
 LIM.flcl.CONV.READ.TEXT.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get wrong  ↩
     BOM in the middle of data #
 LIM.flcl.CONV.READ.TEXT.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15) by  ↩
     line feed (0x25) on input #
 LIM.flcl.CONV.READ.TEXT.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds, filling rest of ↩
      page with blank lines assuming n lines per page [0 = no replacement] #
 LIM.flcl.CONV.READ.TEXT.PADCHAR="" # TYPE: NUMBER HELP: Padding character to be regarded  ↩
     as additional delimiter [0x00] #
 LIM.flcl.CONV.READ.TEXT.DECODE="ALWAYS" # TYPE: NUMBER HELP: Decode encoded data first ( ↩
     remove base encodings) #
 LIM.flcl.CONV.READ.TEXT.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
 LIM.flcl.CONV.READ.TEXT.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password decryption ↩
      #
 LIM.flcl.CONV.READ.TEXT.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to  ↩
     encrypt the FLAMFILE ['********'] #
 LIM.flcl.CONV.READ.TEXT.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management  ↩
     extension (FKME) #
 LIM.flcl.CONV.READ.TEXT.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for  ↩
     FKME ['libfkme'] #
 LIM.flcl.CONV.READ.TEXT.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of  ↩
     FKME ['FKMESTD0'] #
 LIM.flcl.CONV.READ.TEXT.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for the  ↩
     call of FKME [''] #
 LIM.flcl.CONV.READ.TEXT.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character  ↩
```

```
    handling [DETACH] #
 LIM.flcl.CONV.READ.TEXT.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO subsystem ↩
    (only for z/OS) #
 LIM.flcl.CONV.READ.TEXT.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the subsystem ↩
     #
 LIM.flcl.CONV.READ.TEXT.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list for the ↩
    subsystem (comma separated) #
 LIM.flcl.CONV.READ.TEXT.FRCBLK="" # TYPE: SWITCH HELP: Enforce block orientation on record ↩
     oriented devices [FALSE] #
 LIM.flcl.CONV.READ.TEXT.REMOVE="" # TYPE: SWITCH HELP: Remove original file after ↩
    successful processing [FALSE] #
 LIM.flcl.CONV.READ.TEXT.BINERROR="" # TYPE: SWITCH HELP: Enforce an error if binary data ↩
    detected [FALSE] #
 LIM.flcl.CONV.READ.TEXT.CHRERROR="" # TYPE: SWITCH HELP: Enforce an error if character ↩
    data without delimiter detected [FALSE] #
 LIM.flcl.CONV.READ.TEXT.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG ↩
    for read operation #
 LIM.flcl.CONV.READ.TEXT.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable ↩
    FL_PLATFORM for read operation #
 LIM.flcl.CONV.READ.TEXT.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash ↩
    values of input data #
 LIM.flcl.CONV.READ.TEXT.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.CONV.READ.TEXT.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum ↩
    verification #
 LIM.flcl.CONV.READ.TEXT.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string #
 LIM.flcl.CONV.READ.TEXT.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains ↩
    Checksum as hex-string #
 LIM.flcl.CONV.READ.TEXT.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] #
 LIM.flcl.CONV.READ.TEXT.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG] ↩
    #
 LIM.flcl.CONV.READ.TEXT.CHECK="" # TYPE: SWITCH HELP: Activate additional optional ↩
    verifications [FALSE] #
 LIM.flcl.CONV.READ.XML="" # TYPE: OBJECT HELP: Read XML data from a file #

# DESCRIPTION for LIM.flcl.CONV.READ.XML.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.CONV.READ.XML.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read ↩
    access #

# DESCRIPTION for LIM.flcl.CONV.READ.XML.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.
```

```
For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.XML.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.CONV.READ.XML.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.XML.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.READ.XML.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.XML.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional] ↩
      #

# DESCRIPTION for LIM.flcl.CONV.READ.XML.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.XML.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.CONV.READ.XML.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.XML.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.READ.XML.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.XML.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] ↩
      #

# DESCRIPTION for LIM.flcl.CONV.READ.XML.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.XML.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.CONV.READ.XML.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
 LIM.flcl.CONV.READ.XML.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.CONV.READ.XML.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.XML.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] ↩
```

```
     #

 LIM.flcl.CONV.READ.XML.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''==stdin] #
 LIM.flcl.CONV.READ.XML.BLKSIZE="" # TYPE: NUMBER HELP: Block size for read operations [ ↩
    AUTO] #
 LIM.flcl.CONV.READ.XML.NOCMNT="" # TYPE: SWITCH HELP: Ignore XML comments [FALSE] #
 LIM.flcl.CONV.READ.XML.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID [no  ↩
    character conversion, UTF-8/US-ASCII expected] #
 LIM.flcl.CONV.READ.XML.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get wrong  ↩
    BOM in the middle of data #
 LIM.flcl.CONV.READ.XML.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15) by  ↩
    line feed (0x25) on input #
 LIM.flcl.CONV.READ.XML.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds, filling rest of  ↩
    page with blank lines assuming n lines per page [0 = no replacement] #
 LIM.flcl.CONV.READ.XML.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
 LIM.flcl.CONV.READ.XML.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password decryption  ↩
    #
 LIM.flcl.CONV.READ.XML.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to  ↩
    encrypt the FLAMFILE ['********'] #
 LIM.flcl.CONV.READ.XML.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management  ↩
    extension (FKME) #
 LIM.flcl.CONV.READ.XML.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for FKME ↩
     ['libfkme'] #
 LIM.flcl.CONV.READ.XML.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of  ↩
    FKME ['FKMESTD0'] #
 LIM.flcl.CONV.READ.XML.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for the  ↩
    call of FKME [''] #
 LIM.flcl.CONV.READ.XML.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO subsystem ( ↩
    only for z/OS) #
 LIM.flcl.CONV.READ.XML.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the subsystem  ↩
    #
 LIM.flcl.CONV.READ.XML.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list for the  ↩
    subsystem (comma separated) #
 LIM.flcl.CONV.READ.XML.FRCBLK="" # TYPE: SWITCH HELP: Enforce block orientation on record  ↩
    oriented devices [FALSE] #
 LIM.flcl.CONV.READ.XML.REMOVE="" # TYPE: SWITCH HELP: Remove original file after  ↩
    successful processing [FALSE] #
 LIM.flcl.CONV.READ.XML.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG for ↩
     read operation #
 LIM.flcl.CONV.READ.XML.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
    FL_PLATFORM for read operation #
 LIM.flcl.CONV.READ.XML.HASH="" # TYPE: OBJECT HELP: Generation or check  of one way hash  ↩
    values of input data #
 LIM.flcl.CONV.READ.XML.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.CONV.READ.XML.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
 LIM.flcl.CONV.READ.XML.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string #
 LIM.flcl.CONV.READ.XML.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
 LIM.flcl.CONV.READ.XML.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] #
 LIM.flcl.CONV.READ.XML.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG] #
 LIM.flcl.CONV.READ.XML.CHECK="" # TYPE: SWITCH HELP: Activate additional optional  ↩
    verifications [FALSE] #
 LIM.flcl.CONV.READ.RECORD="" # TYPE: OBJECT HELP: Read records from a file #

# DESCRIPTION for LIM.flcl.CONV.READ.RECORD.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.
```

```
 #
 LIM.flcl.CONV.READ.RECORD.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read  ↩
     access #

# DESCRIPTION for LIM.flcl.CONV.READ.RECORD.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.RECORD.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.CONV.READ.RECORD.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.RECORD.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.READ.RECORD.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.RECORD.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.READ.RECORD.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.RECORD.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.CONV.READ.RECORD.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.RECORD.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.READ.RECORD.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.RECORD.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #
```

```
# DESCRIPTION for LIM.flcl.CONV.READ.RECORD.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.RECORD.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.CONV.READ.RECORD.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] ↩
      #
 LIM.flcl.CONV.READ.RECORD.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.CONV.READ.RECORD.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.RECORD.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
    optional] #

 LIM.flcl.CONV.READ.RECORD.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''==stdin ↩
    ] #
 LIM.flcl.CONV.READ.RECORD.RECMODE="" # TYPE: NUMBER HELP: Mode used to read records [STOP] ↩
      #
 LIM.flcl.CONV.READ.RECORD.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character  ↩
    handling [DETACH] #
 LIM.flcl.CONV.READ.RECORD.SUPPADDING="" # TYPE: SWITCH HELP: Suppress trailing padding  ↩
    characters (useful for fix records) #
 LIM.flcl.CONV.READ.RECORD.ORGANIZATION="" # TYPE: NUMBER HELP: File organization [catalog/ ↩
    SEQ] #
 LIM.flcl.CONV.READ.RECORD.BLKSIZE="" # TYPE: NUMBER HELP: Block size for read operations [ ↩
    catalog/AUTO] #
 LIM.flcl.CONV.READ.RECORD.RECFORMAT="" # TYPE: NUMBER HELP: Record format for read  ↩
    operation [catalog/VAR] #
 LIM.flcl.CONV.READ.RECORD.RECLENGTH="" # TYPE: NUMBER HELP: Maximal record length for read ↩
     operation [catalog/512] #
 LIM.flcl.CONV.READ.RECORD.RECDELIM="" # TYPE: STRING HELP: Delimiter used to define end of ↩
     record (if RECF=DLM used) #
 LIM.flcl.CONV.READ.RECORD.RECCOUNT="" # TYPE: NUMBER HELP: Amount of records handled in  ↩
    one read operation [catalog/128] #
 LIM.flcl.CONV.READ.RECORD.LENFORMAT="" # TYPE: OVERLAY HELP: Format of the length field  ↩
    for open platforms [word] #
 LIM.flcl.CONV.READ.RECORD.LENFORMAT.INTEGER="" # TYPE: OBJECT HELP: Use integer format for ↩
     the length field #
 LIM.flcl.CONV.READ.RECORD.LENFORMAT.INTEGER.ENDIAN="" # TYPE: NUMBER HELP: Byte order [ ↩
    SYSTEM] #
 LIM.flcl.CONV.READ.RECORD.LENFORMAT.INTEGER.WIDTH="" # TYPE: NUMBER HELP: Bit width/rate [ ↩
    B32] #
 LIM.flcl.CONV.READ.RECORD.LENFORMAT.INTEGER.INCLUDE="" # TYPE: SWITCH HELP: Include length ↩
     field in length value [OFF] #
 LIM.flcl.CONV.READ.RECORD.LENFORMAT.STRING="" # TYPE: OBJECT HELP: Use string format for  ↩
    the length field #
 LIM.flcl.CONV.READ.RECORD.LENFORMAT.STRING.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
 LIM.flcl.CONV.READ.RECORD.LENFORMAT.STRING.COUNT="" # TYPE: NUMBER HELP: Character count [ ↩
    C06] #
 LIM.flcl.CONV.READ.RECORD.LENFORMAT.STRING.BASE="" # TYPE: NUMBER HELP: String base [B10]  ↩
    #
```

```
LIM.flcl.CONV.READ.RECORD.LENFORMAT.STRING.INCLUDE="" # TYPE: SWITCH HELP: Include length field in length value [OFF] #
LIM.flcl.CONV.READ.RECORD.LENFORMAT.BCD="" # TYPE: OBJECT HELP: Use BCD format for the length field #
LIM.flcl.CONV.READ.RECORD.LENFORMAT.BCD.WIDTH="" # TYPE: NUMBER HELP: Bit width/rate [B32] #
LIM.flcl.CONV.READ.RECORD.LENFORMAT.BCD.INCLUDE="" # TYPE: SWITCH HELP: Include length field in length value [OFF] #
LIM.flcl.CONV.READ.RECORD.LENFORMAT.HOST="" # TYPE: OBJECT HELP: Use host format (LLxx) for the length field #
LIM.flcl.CONV.READ.RECORD.LENFORMAT.HOST.ENDIAN="" # TYPE: NUMBER HELP: Byte order [BIG ENDIAN] for first 16 Bit #
LIM.flcl.CONV.READ.RECORD.LENFORMAT.HOST.EXCLUDE="" # TYPE: SWITCH HELP: Record length field not part of the length value [OFF] #
LIM.flcl.CONV.READ.RECORD.KEYPOSITION="" # TYPE: NUMBER HELP: Key position (>=1) for index/key sequential data sets [0=NONE] #
LIM.flcl.CONV.READ.RECORD.KEYLENGTH="" # TYPE: NUMBER HELP: Key length (>=1) for index/key sequential data sets [if KEYPOS>0 then 8] #
LIM.flcl.CONV.READ.RECORD.DECODE="" # TYPE: NUMBER HELP: Decode encoded, encrypted or compressed data [0=NONE] #
LIM.flcl.CONV.READ.RECORD.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
LIM.flcl.CONV.READ.RECORD.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password decryption #
LIM.flcl.CONV.READ.RECORD.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to encrypt the FLAMFILE ['********'] #
LIM.flcl.CONV.READ.RECORD.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management extension (FKME) #
LIM.flcl.CONV.READ.RECORD.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for FKME ['libfkme'] #
LIM.flcl.CONV.READ.RECORD.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of FKME ['FKMESTD0'] #
LIM.flcl.CONV.READ.RECORD.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for the call of FKME [''] #
LIM.flcl.CONV.READ.RECORD.FDECODE="" # TYPE: OVERLAY HELP: Additional enforced decoding [NONE] #
LIM.flcl.CONV.READ.RECORD.FDECODE.BASE64="" # TYPE: SWITCH HELP: Base64 decoding (RFC4648) #
LIM.flcl.CONV.READ.RECORD.FDECODE.BASE32="" # TYPE: SWITCH HELP: Base32 decoding (RFC4648) #
LIM.flcl.CONV.READ.RECORD.FDECODE.BASE16="" # TYPE: SWITCH HELP: Base16 decoding (RFC4648) #
LIM.flcl.CONV.READ.RECORD.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID ( enables text handling if set) #
LIM.flcl.CONV.READ.RECORD.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get wrong BOM in the middle of data #
LIM.flcl.CONV.READ.RECORD.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO subsystem (only for z/OS) #
LIM.flcl.CONV.READ.RECORD.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the subsystem #
LIM.flcl.CONV.READ.RECORD.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list for the subsystem (comma separated) #
LIM.flcl.CONV.READ.RECORD.REMOVE="" # TYPE: SWITCH HELP: Remove original file after successful processing [FALSE] #
LIM.flcl.CONV.READ.RECORD.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG for read operation #
LIM.flcl.CONV.READ.RECORD.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable FL_PLATFORM for read operation #
LIM.flcl.CONV.READ.RECORD.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash values of input data #
LIM.flcl.CONV.READ.RECORD.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
LIM.flcl.CONV.READ.RECORD.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum verification #
```

```
 LIM.flcl.CONV.READ.RECORD.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string  ↩
     #
 LIM.flcl.CONV.READ.RECORD.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
     Checksum as hex-string #
 LIM.flcl.CONV.READ.RECORD.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] ↩
      #
 LIM.flcl.CONV.READ.RECORD.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG ↩
     ] #
 LIM.flcl.CONV.READ.RECORD.CHECK="" # TYPE: SWITCH HELP: Activate additional optional  ↩
     verifications [FALSE] #
 LIM.flcl.CONV.READ.FLAM4="" # TYPE: OBJECT HELP: Read records from a FLAM4FILE #

# DESCRIPTION for LIM.flcl.CONV.READ.FLAM4.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.CONV.READ.FLAM4.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read  ↩
     access #

# DESCRIPTION for LIM.flcl.CONV.READ.FLAM4.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.FLAM4.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.CONV.READ.FLAM4.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.FLAM4.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.READ.FLAM4.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.FLAM4.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.READ.FLAM4.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.
```

```
To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.FLAM4.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.CONV.READ.FLAM4.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.FLAM4.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.READ.FLAM4.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.FLAM4.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.READ.FLAM4.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.FLAM4.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.CONV.READ.FLAM4.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""]  ↩
     #
 LIM.flcl.CONV.READ.FLAM4.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.CONV.READ.FLAM4.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.FLAM4.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.CONV.READ.FLAM4.FILE="" # TYPE: STRING HELP: Name/URL of the FLAMFILE to read [ ↩
     required] #
 LIM.flcl.CONV.READ.FLAM4.MEMBER="" # TYPE: STRING HELP: Name or Index of the member in the ↩
      FLAMFILE to read #
 LIM.flcl.CONV.READ.FLAM4.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character  ↩
     handling [DETACH] #
 LIM.flcl.CONV.READ.FLAM4.RECCOUNT="" # TYPE: NUMBER HELP: Amount of records handled in one ↩
      read operation [128] #
 LIM.flcl.CONV.READ.FLAM4.RECLENGTH="" # TYPE: NUMBER HELP: Maximal record length for read  ↩
     operation [512] #
 LIM.flcl.CONV.READ.FLAM4.TRUNCATE="" # TYPE: SWITCH HELP: Allow truncation of records [ ↩
     FALSE] #
 LIM.flcl.CONV.READ.FLAM4.PASSWORD="" # TYPE: STRING HELP: Passphrase to decrypt the  ↩
     FLAMFILE [''] #
 LIM.flcl.CONV.READ.FLAM4.KMLIBRARY="" # TYPE: STRING HELP: Library name for FKME [''] #
 LIM.flcl.CONV.READ.FLAM4.KMFUNCTION="" # TYPE: STRING HELP: Function name of FKME [''] #
 LIM.flcl.CONV.READ.FLAM4.KMPARAMETER="" # TYPE: STRING HELP: Parameter for the call of  ↩
     FKME [''] #
 LIM.flcl.CONV.READ.FLAM4.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID (enables ↩
```

```
     text handling if set) #
 LIM.flcl.CONV.READ.FLAM4.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get wrong ↩
     BOM in the middle of data #
 LIM.flcl.CONV.READ.FLAM4.REMOVE="" # TYPE: SWITCH HELP: Remove FLAMFILE after successful  ↩
    processing [FALSE] #
 LIM.flcl.CONV.READ.FLAM4.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG  ↩
    for read operation #
 LIM.flcl.CONV.READ.FLAM4.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
    FL_PLATFORM for read operation #
 LIM.flcl.CONV.READ.FLAM4.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash  ↩
    values of input data #
 LIM.flcl.CONV.READ.FLAM4.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.CONV.READ.FLAM4.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
 LIM.flcl.CONV.READ.FLAM4.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string #
 LIM.flcl.CONV.READ.FLAM4.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
 LIM.flcl.CONV.READ.FLAM4.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU]  ↩
    #
 LIM.flcl.CONV.READ.FLAM4.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG] ↩
     #
 LIM.flcl.CONV.READ.AUTO="" # TYPE: OBJECT HELP: Read binary, text or XML data record- or  ↩
    block-oriented #

# DESCRIPTION for LIM.flcl.CONV.READ.AUTO.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.CONV.READ.AUTO.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read  ↩
    access #

# DESCRIPTION for LIM.flcl.CONV.READ.AUTO.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.AUTO.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.CONV.READ.AUTO.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.AUTO.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.READ.AUTO.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
```

```
 LIM.flcl.CONV.READ.AUTO.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

# DESCRIPTION for LIM.flcl.CONV.READ.AUTO.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.AUTO.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.CONV.READ.AUTO.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.READ.AUTO.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.READ.AUTO.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.AUTO.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

# DESCRIPTION for LIM.flcl.CONV.READ.AUTO.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.READ.AUTO.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.CONV.READ.AUTO.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
 LIM.flcl.CONV.READ.AUTO.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.CONV.READ.AUTO.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.READ.AUTO.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

 LIM.flcl.CONV.READ.AUTO.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''==stdin]  ↩
     #
 LIM.flcl.CONV.READ.AUTO.BLKSIZE="" # TYPE: NUMBER HELP: Block size to read [AUTO] #
 LIM.flcl.CONV.READ.AUTO.RECLENGTH="" # TYPE: NUMBER HELP: Maximum record length (Host) /  ↩
     line length (Unix/Win) for text parsing [512] #
 LIM.flcl.CONV.READ.AUTO.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds, filling rest of ↩
      page with blank lines assuming n lines per page [0 = no replacement] #
 LIM.flcl.CONV.READ.AUTO.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID [auto  ↩
```

```
    detection] #
 LIM.flcl.CONV.READ.AUTO.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get wrong ↩
    BOM in the middle of data #
 LIM.flcl.CONV.READ.AUTO.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15) by ↩
    line feed (0x25) on input #
 LIM.flcl.CONV.READ.AUTO.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
 LIM.flcl.CONV.READ.AUTO.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password decryption ↩
     #
 LIM.flcl.CONV.READ.AUTO.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to ↩
    encrypt the FLAMFILE ['********'] #
 LIM.flcl.CONV.READ.AUTO.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management ↩
    extension (FKME) #
 LIM.flcl.CONV.READ.AUTO.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for ↩
    FKME ['libfkme'] #
 LIM.flcl.CONV.READ.AUTO.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of ↩
    FKME ['FKMESTD0'] #
 LIM.flcl.CONV.READ.AUTO.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for the ↩
    call of FKME [''] #
 LIM.flcl.CONV.READ.AUTO.REMOVE="" # TYPE: SWITCH HELP: Remove original file after ↩
    successful processing [FALSE] #
 LIM.flcl.CONV.READ.AUTO.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG ↩
    for read operation #
 LIM.flcl.CONV.READ.AUTO.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable ↩
    FL_PLATFORM for read operation #
 LIM.flcl.CONV.READ.AUTO.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash ↩
    values of input data #
 LIM.flcl.CONV.READ.AUTO.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.CONV.READ.AUTO.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum ↩
    verification #
 LIM.flcl.CONV.READ.AUTO.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string #
 LIM.flcl.CONV.READ.AUTO.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains ↩
    Checksum as hex-string #
 LIM.flcl.CONV.READ.AUTO.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] #
 LIM.flcl.CONV.READ.AUTO.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG] ↩
    #
 LIM.flcl.CONV.READ.AUTO.CHECK="" # TYPE: SWITCH HELP: Activate additional optional ↩
    verifications [FALSE] #
 LIM.flcl.CONV.READ.FILE="" # TYPE: STRING HELP: Abbreviation for read.auto(file=...) #
 LIM.flcl.CONV.WRITE="" # TYPE: OVERLAY HELP: Outbound parameter to write data #
 LIM.flcl.CONV.WRITE.BINARY="" # TYPE: OBJECT HELP: Write binary data to a file #

# DESCRIPTION for LIM.flcl.CONV.WRITE.BINARY.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.CONV.WRITE.BINARY.NET="" # TYPE: OVERLAY HELP: Network parameter for remote write ↩
     access #

# DESCRIPTION for LIM.flcl.CONV.WRITE.BINARY.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.
```

```
To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.BINARY.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.CONV.WRITE.BINARY.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.WRITE.BINARY.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.WRITE.BINARY.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.BINARY.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.WRITE.BINARY.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.BINARY.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.CONV.WRITE.BINARY.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.WRITE.BINARY.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.WRITE.BINARY.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.BINARY.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.WRITE.BINARY.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.
```

```
#
LIM.flcl.CONV.WRITE.BINARY.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

LIM.flcl.CONV.WRITE.BINARY.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name  ↩
    [""] #
LIM.flcl.CONV.WRITE.BINARY.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
LIM.flcl.CONV.WRITE.BINARY.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
LIM.flcl.CONV.WRITE.BINARY.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
    optional] #

LIM.flcl.CONV.WRITE.BINARY.FILE="" # TYPE: STRING HELP: Name/URL of file to write [''== ↩
    origin.ext] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC="" # TYPE: OBJECT HELP: Platform dependent file  ↩
    allocation parameter #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.ORGANIZATION="" # TYPE: NUMBER HELP: File organization [ ↩
    SEQ] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.BLKSIZE="" # TYPE: NUMBER HELP: Block size [system] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.RECFORMAT="" # TYPE: NUMBER HELP: Record format [auto] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.RECLENGTH="" # TYPE: NUMBER HELP: Record length (data  ↩
    only, no print control and length fields) #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.KEYDISP="" # TYPE: NUMBER HELP: Key disposition for  ↩
    index/key sequential data sets [OLD] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.KEYPOSITION="" # TYPE: NUMBER HELP: Key position (>=1)  ↩
    for index/key sequential data sets [1] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.KEYLENGTH="" # TYPE: NUMBER HELP: Key length (>=1) for  ↩
    index/key sequential data sets [8] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO  ↩
    subsystem (only for z/OS) #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the ↩
     subsystem #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list  ↩
    for the subsystem (comma separated) #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE="" # TYPE: OBJECT HELP: Platform dependent space  ↩
    definition parameter #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.PRIMARY="" # TYPE: NUMBER HELP: Primary size in  ↩
    megabytes #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.SECONDARY="" # TYPE: NUMBER HELP: Secondary size  ↩
    in megabytes #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.DIRBLOCKS="" # TYPE: NUMBER HELP: Directory blocks ↩
     for partitioned data sets #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.VOLSER="" # TYPE: STRING HELP: Volume serial  ↩
    number #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.VOLUNIT="" # TYPE: STRING HELP: Volume unit #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.DATACLASS="" # TYPE: STRING HELP: Data class #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.MGMTCLASS="" # TYPE: STRING HELP: Management class ↩
     #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.STORCLASS="" # TYPE: STRING HELP: Storage class #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.NORMALDISP="" # TYPE: NUMBER HELP: Close  ↩
    disposition for normal end [CATALOG] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.ABORTDISP="" # TYPE: NUMBER HELP: Close  ↩
    disposition for abnormal end [DELETE] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.RELEASE="" # TYPE: NUMBER HELP: Release unused  ↩
    space when file is closed [SEQ=YES, PDS=NO] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.UNMOVABLE="" # TYPE: SWITCH HELP: Allocate space  ↩
    unmovable [FALSE] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.CONTIG="" # TYPE: SWITCH HELP: Allocate space  ↩
    contiguously [FALSE] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.PERMANENT="" # TYPE: SWITCH HELP: Allocate space  ↩
    permanently [FALSE] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.SPACE.RLS="" # TYPE: NUMBER HELP: Specifies the type of  ↩
    record level sharing (RLS) being done for a specific data set [NONE] #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.LIKE="" # TYPE: STRING HELP: Specifies the name of the  ↩
```

```
    model data set from which the attributes are to be copied #
LIM.flcl.CONV.WRITE.BINARY.FALLOC.RENEW="" # TYPE: SWITCH HELP: Enforce new allocation by  ↩
    a remove of an existing file #
LIM.flcl.CONV.WRITE.BINARY.REMAIN="" # TYPE: SWITCH HELP: Remain existing files (don't  ↩
    overwrite) [FALSE] #
LIM.flcl.CONV.WRITE.BINARY.APPEND="" # TYPE: SWITCH HELP: Append data to this file [FALSE] ↩
     #
LIM.flcl.CONV.WRITE.BINARY.FLUSH="" # TYPE: SWITCH HELP: Enforce flush in front of close  ↩
    file [FALSE] #
LIM.flcl.CONV.WRITE.BINARY.METHOD="" # TYPE: NUMBER HELP: Method to convert records in  ↩
    blocks [NONE] #
LIM.flcl.CONV.WRITE.BINARY.COMPRESS="" # TYPE: OVERLAY HELP: Compression procedure [NONE]  ↩
    #
LIM.flcl.CONV.WRITE.BINARY.COMPRESS.GZIP="" # TYPE: OBJECT HELP: Compress data compliant  ↩
    to RCF1952 (GZIP) #
LIM.flcl.CONV.WRITE.BINARY.COMPRESS.GZIP.LEVEL="" # TYPE: NUMBER HELP: Compression level [ ↩
    AUTO==6] #
LIM.flcl.CONV.WRITE.BINARY.COMPRESS.GZIP.COMMENT="" # TYPE: STRING HELP: Comment for  ↩
    header [NONE] #
LIM.flcl.CONV.WRITE.BINARY.COMPRESS.GZIP.CHECK="" # TYPE: SWITCH HELP: Calculate checksum  ↩
    over GZIP header [FALSE] #
LIM.flcl.CONV.WRITE.BINARY.COMPRESS.BZIP2="" # TYPE: OBJECT HELP: BZIP2 data compression  ↩
    technique #
LIM.flcl.CONV.WRITE.BINARY.COMPRESS.BZIP2.LEVEL="" # TYPE: NUMBER HELP: Compression level  ↩
    (defines block size) [AUTO] #
LIM.flcl.CONV.WRITE.BINARY.COMPRESS.XZ="" # TYPE: OBJECT HELP: XZ (LZMA) data compression  ↩
    technique #
LIM.flcl.CONV.WRITE.BINARY.COMPRESS.XZ.LEVEL="" # TYPE: NUMBER HELP: Compression level [ ↩
    AUTO==6] #
LIM.flcl.CONV.WRITE.BINARY.COMPRESS.FLAM4="" # TYPE: OBJECT HELP: Compress data with FLAM  ↩
    #
LIM.flcl.CONV.WRITE.BINARY.COMPRESS.FLAM4.MODE="" # TYPE: NUMBER HELP: Compression mode [ ↩
    ADC] #
LIM.flcl.CONV.WRITE.BINARY.COMPRESS.FLAM4.COMMENT="" # TYPE: STRING HELP: Comment for the  ↩
    member in the FLAMFILE [internal use] #
LIM.flcl.CONV.WRITE.BINARY.ENCRYPT="" # TYPE: OVERLAY HELP: Encryption procedure [NONE] #
LIM.flcl.CONV.WRITE.BINARY.ENCRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password  ↩
    encryption #
LIM.flcl.CONV.WRITE.BINARY.ENCRYPT.F4PWD.MODE="" # TYPE: NUMBER HELP: Encryption mode [AES ↩
    ] #
LIM.flcl.CONV.WRITE.BINARY.ENCRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to  ↩
    encrypt the FLAMFILE ['********'] #
LIM.flcl.CONV.WRITE.BINARY.ENCRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management  ↩
    extension (FKME) #
LIM.flcl.CONV.WRITE.BINARY.ENCRYPT.F4KME.MODE="" # TYPE: NUMBER HELP: Encryption mode [AES ↩
    ] #
LIM.flcl.CONV.WRITE.BINARY.ENCRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for  ↩
    FKME ['libfkme'] #
LIM.flcl.CONV.WRITE.BINARY.ENCRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of ↩
     FKME ['FKMESTD0'] #
LIM.flcl.CONV.WRITE.BINARY.ENCRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for  ↩
    the call of FKME [''] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE="" # TYPE: OVERLAY HELP: Encoding procedure [NONE] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE64="" # TYPE: OBJECT HELP: Base64 encoding (RFC4648) ↩
     #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE64.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE64.MODE="" # TYPE: NUMBER HELP: Encoding mode [BLOCK ↩
    ] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE64.LINE="" # TYPE: NUMBER HELP: Maximum length of an ↩
     encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE64.DELIM="" # TYPE: NUMBER HELP: Line delimiter to  ↩
```

```
    use when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE64.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC checksum ↩
     to the base encoded data [FALSE] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE64.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR  ↩
    encoding and defines corresponding parameter #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE64.ARMOR.HEADER="" # TYPE: STRING HELP: String for  ↩
    the header [AUTO] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE64.ARMOR.VERSION="" # TYPE: STRING HELP: Version  ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE64.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment  ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE32="" # TYPE: OBJECT HELP: Base32 encoding (RFC4648) ↩
     #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE32.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE32.MODE="" # TYPE: NUMBER HELP: Encoding mode [BLOCK ↩
    ] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE32.LINE="" # TYPE: NUMBER HELP: Maximum length of an ↩
     encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE32.DELIM="" # TYPE: NUMBER HELP: Line delimiter to  ↩
    use when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE32.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC checksum ↩
     to the base encoded data [FALSE] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE32.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR  ↩
    encoding and defines corresponding parameter #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE32.ARMOR.HEADER="" # TYPE: STRING HELP: String for  ↩
    the header [AUTO] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE32.ARMOR.VERSION="" # TYPE: STRING HELP: Version  ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE32.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment  ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE16="" # TYPE: OBJECT HELP: Base16 encoding (RFC4648) ↩
     #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE16.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE16.MODE="" # TYPE: NUMBER HELP: Encoding mode [BLOCK ↩
    ] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE16.LINE="" # TYPE: NUMBER HELP: Maximum length of an ↩
     encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE16.DELIM="" # TYPE: NUMBER HELP: Line delimiter to  ↩
    use when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE16.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC checksum ↩
     to the base encoded data [FALSE] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE16.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR  ↩
    encoding and defines corresponding parameter #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE16.ARMOR.HEADER="" # TYPE: STRING HELP: String for  ↩
    the header [AUTO] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE16.ARMOR.VERSION="" # TYPE: STRING HELP: Version  ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.BASE16.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment  ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.OPGP="" # TYPE: OBJECT HELP: Base64 encoding with 76  ↩
    character, CRC checksum and Armor clasp (RFC4880) #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.OPGP.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
LIM.flcl.CONV.WRITE.BINARY.ENCODE.OPGP.DELIM="" # TYPE: NUMBER HELP: Line delimiter to use ↩
     when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.BINARY.FLAM4NDC="" # TYPE: SWITCH HELP: Write data as FLAM4FILE in  ↩
    MODE=NDC(No Data Compression) [FALSE] #
LIM.flcl.CONV.WRITE.BINARY.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character  ↩
    handling [DETACH] #
LIM.flcl.CONV.WRITE.BINARY.FRCREC="" # TYPE: SWITCH HELP: Enforce record orientation on  ↩
```

```
    record oriented devices [FALSE] #
 LIM.flcl.CONV.WRITE.BINARY.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
    FL_PLATFORM for write operation #
 LIM.flcl.CONV.WRITE.BINARY.HASH="" # TYPE: OBJECT HELP: Generation or check of one way  ↩
    hash values of output data #
 LIM.flcl.CONV.WRITE.BINARY.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.CONV.WRITE.BINARY.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
 LIM.flcl.CONV.WRITE.BINARY.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string ↩
     #
 LIM.flcl.CONV.WRITE.BINARY.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
 LIM.flcl.CONV.WRITE.BINARY.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU ↩
    ] #
 LIM.flcl.CONV.WRITE.BINARY.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [ ↩
    LOG] #
 LIM.flcl.CONV.WRITE.CHARACTER="" # TYPE: OBJECT HELP: Write character data to a file #

# DESCRIPTION for LIM.flcl.CONV.WRITE.CHARACTER.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.CONV.WRITE.CHARACTER.NET="" # TYPE: OVERLAY HELP: Network parameter for remote  ↩
    write access #

# DESCRIPTION for LIM.flcl.CONV.WRITE.CHARACTER.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.CHARACTER.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.CONV.WRITE.CHARACTER.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost]  ↩
    #
 LIM.flcl.CONV.WRITE.CHARACTER.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.WRITE.CHARACTER.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.CHARACTER.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
    optional] #

# DESCRIPTION for LIM.flcl.CONV.WRITE.CHARACTER.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
```

```
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.CHARACTER.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.CONV.WRITE.CHARACTER.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost]  ↩
     #
 LIM.flcl.CONV.WRITE.CHARACTER.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.WRITE.CHARACTER.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.CHARACTER.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.WRITE.CHARACTER.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.CHARACTER.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.CONV.WRITE.CHARACTER.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name  ↩
     [""] #
 LIM.flcl.CONV.WRITE.CHARACTER.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.CONV.WRITE.CHARACTER.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.CHARACTER.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.CONV.WRITE.CHARACTER.FILE="" # TYPE: STRING HELP: Name/URL of file to write [''== ↩
     origin.ext] #
 LIM.flcl.CONV.WRITE.CHARACTER.FALLOC="" # TYPE: OBJECT HELP: Platform dependent file  ↩
     allocation parameter #
 LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.ORGANIZATION="" # TYPE: NUMBER HELP: File  ↩
     organization [SEQ] #
 LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.BLKSIZE="" # TYPE: NUMBER HELP: Block size [system] #
 LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.RECFORMAT="" # TYPE: NUMBER HELP: Record format [auto ↩
     ] #
 LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.RECLENGTH="" # TYPE: NUMBER HELP: Record length (data ↩
      only, no print control and length fields) #
```

```
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.KEYDISP="" # TYPE: NUMBER HELP: Key disposition for index/key sequential data sets [OLD] #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.KEYPOSITION="" # TYPE: NUMBER HELP: Key position (>=1) for index/key sequential data sets [1] #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.KEYLENGTH="" # TYPE: NUMBER HELP: Key length (>=1) for index/key sequential data sets [8] #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO subsystem (only for z/OS) #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the subsystem #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SUBSYSTEM.PARM="'" # TYPE: STRING HELP: Parameter list for the subsystem (comma separated) #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE="" # TYPE: OBJECT HELP: Platform dependent space definition parameter #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.PRIMARY="" # TYPE: NUMBER HELP: Primary size in megabytes #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.SECONDARY="" # TYPE: NUMBER HELP: Secondary size in megabytes #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.DIRBLOCKS="" # TYPE: NUMBER HELP: Directory blocks for partitioned data sets #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.VOLSER="" # TYPE: STRING HELP: Volume serial number #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.VOLUNIT="" # TYPE: STRING HELP: Volume unit #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.DATACLASS="" # TYPE: STRING HELP: Data class #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.MGMTCLASS="" # TYPE: STRING HELP: Management class #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.STORCLASS="" # TYPE: STRING HELP: Storage class #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.NORMALDISP="" # TYPE: NUMBER HELP: Close disposition for normal end [CATALOG] #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.ABORTDISP="" # TYPE: NUMBER HELP: Close disposition for abnormal end [DELETE] #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.RELEASE="" # TYPE: NUMBER HELP: Release unused space when file is closed [SEQ=YES, PDS=NO] #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.UNMOVABLE="" # TYPE: SWITCH HELP: Allocate space unmovable [FALSE] #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.CONTIG="" # TYPE: SWITCH HELP: Allocate space contiguously [FALSE] #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.PERMANENT="" # TYPE: SWITCH HELP: Allocate space permanently [FALSE] #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.SPACE.RLS="" # TYPE: NUMBER HELP: Specifies the type of record level sharing (RLS) being done for a specific data set [NONE] #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.LIKE="" # TYPE: STRING HELP: Specifies the name of the model data set from which the attributes are to be copied #
LIM.flcl.CONV.WRITE.CHARACTER.FALLOC.RENEW="" # TYPE: SWITCH HELP: Enforce new allocation by a remove of an existing file #
LIM.flcl.CONV.WRITE.CHARACTER.REMAIN="" # TYPE: SWITCH HELP: Remain existing files (don't overwrite) [FALSE] #
LIM.flcl.CONV.WRITE.CHARACTER.APPEND="" # TYPE: SWITCH HELP: Append data to this file [ FALSE] #
LIM.flcl.CONV.WRITE.CHARACTER.FLUSH="" # TYPE: SWITCH HELP: Enforce flush in front of close file [FALSE] #
LIM.flcl.CONV.WRITE.CHARACTER.METHOD="" # TYPE: NUMBER HELP: Method to convert records in blocks [NONE] #
LIM.flcl.CONV.WRITE.CHARACTER.CCSID="" # TYPE: STRING HELP: Conversion to this CCSID [ DEFAULT] #
LIM.flcl.CONV.WRITE.CHARACTER.CASE="" # TYPE: NUMBER HELP: Define upper, lower or special case conversion [NONE] #
LIM.flcl.CONV.WRITE.CHARACTER.BOM="" # TYPE: SWITCH HELP: Write byte order mark in front of the output text [FALSE] #
LIM.flcl.CONV.WRITE.CHARACTER.CHRMODE="" # TYPE: NUMBER HELP: Define handling of non- convertible characters [STOP] #
```

```
LIM.flcl.CONV.WRITE.CHARACTER.ELF2NL="" # TYPE: SWITCH HELP: Replace EBCDIC line feed (0 ↩
    x25) by new line (0x15) on output #
LIM.flcl.CONV.WRITE.CHARACTER.SUBCHAR="" # TYPE: NUMBER HELP: Default substitution  ↩
    character list (maximal 8 code points #
LIM.flcl.CONV.WRITE.CHARACTER.SYSTABLE="" # TYPE: NUMBER HELP: Define a preloaded system  ↩
    substitution table #
LIM.flcl.CONV.WRITE.CHARACTER.USRTABLE="" # TYPE: STRING HELP: Name of the file containing ↩
     the user conversion table #
LIM.flcl.CONV.WRITE.CHARACTER.REPORTFILE="" # TYPE: STRING HELP: Name of the report file  ↩
    to log substitutions #
LIM.flcl.CONV.WRITE.CHARACTER.COMPRESS="" # TYPE: OVERLAY HELP: Compression procedure [ ↩
    NONE] #
LIM.flcl.CONV.WRITE.CHARACTER.COMPRESS.GZIP="" # TYPE: OBJECT HELP: Compress data  ↩
    compliant to RCF1952 (GZIP) #
LIM.flcl.CONV.WRITE.CHARACTER.COMPRESS.GZIP.LEVEL="" # TYPE: NUMBER HELP: Compression  ↩
    level [AUTO==6] #
LIM.flcl.CONV.WRITE.CHARACTER.COMPRESS.GZIP.COMMENT="" # TYPE: STRING HELP: Comment for  ↩
    header [NONE] #
LIM.flcl.CONV.WRITE.CHARACTER.COMPRESS.GZIP.CHECK="" # TYPE: SWITCH HELP: Calculate  ↩
    checksum over GZIP header [FALSE] #
LIM.flcl.CONV.WRITE.CHARACTER.COMPRESS.BZIP2="" # TYPE: OBJECT HELP: BZIP2 data  ↩
    compression technique #
LIM.flcl.CONV.WRITE.CHARACTER.COMPRESS.BZIP2.LEVEL="" # TYPE: NUMBER HELP: Compression  ↩
    level (defines block size) [AUTO] #
LIM.flcl.CONV.WRITE.CHARACTER.COMPRESS.XZ="" # TYPE: OBJECT HELP: XZ (LZMA) data  ↩
    compression technique #
LIM.flcl.CONV.WRITE.CHARACTER.COMPRESS.XZ.LEVEL="" # TYPE: NUMBER HELP: Compression level  ↩
    [AUTO==6] #
LIM.flcl.CONV.WRITE.CHARACTER.COMPRESS.FLAM4="" # TYPE: OBJECT HELP: Compress data with  ↩
    FLAM #
LIM.flcl.CONV.WRITE.CHARACTER.COMPRESS.FLAM4.MODE="" # TYPE: NUMBER HELP: Compression mode ↩
     [ADC] #
LIM.flcl.CONV.WRITE.CHARACTER.COMPRESS.FLAM4.COMMENT="" # TYPE: STRING HELP: Comment for  ↩
    the member in the FLAMFILE [internal use] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCRYPT="" # TYPE: OVERLAY HELP: Encryption procedure [NONE] ↩
     #
LIM.flcl.CONV.WRITE.CHARACTER.ENCRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password  ↩
    encryption #
LIM.flcl.CONV.WRITE.CHARACTER.ENCRYPT.F4PWD.MODE="" # TYPE: NUMBER HELP: Encryption mode [ ↩
    AES] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to ↩
     encrypt the FLAMFILE ['********'] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key  ↩
    management extension (FKME) #
LIM.flcl.CONV.WRITE.CHARACTER.ENCRYPT.F4KME.MODE="" # TYPE: NUMBER HELP: Encryption mode [ ↩
    AES] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name  ↩
    for FKME ['libfkme'] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name ↩
     of FKME ['FKMESTD0'] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter  ↩
    for the call of FKME [''] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE="" # TYPE: OVERLAY HELP: Encoding procedure [NONE] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE64="" # TYPE: OBJECT HELP: Base64 encoding ( ↩
    RFC4648) #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE64.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE64.MODE="" # TYPE: NUMBER HELP: Encoding mode [ ↩
    BLOCK] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE64.LINE="" # TYPE: NUMBER HELP: Maximum length of ↩
     an encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE64.DELIM="" # TYPE: NUMBER HELP: Line delimiter  ↩
```

```
    to use when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE64.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC ↩
    checksum to the base encoded data [FALSE] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE64.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR ↩
    encoding and defines corresponding parameter #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE64.ARMOR.HEADER="" # TYPE: STRING HELP: String ↩
    for the header [AUTO] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE64.ARMOR.VERSION="" # TYPE: STRING HELP: Version ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE64.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE32="" # TYPE: OBJECT HELP: Base32 encoding ( ↩
    RFC4648) #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE32.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE32.MODE="" # TYPE: NUMBER HELP: Encoding mode [ ↩
    BLOCK] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE32.LINE="" # TYPE: NUMBER HELP: Maximum length of ↩
     an encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE32.DELIM="" # TYPE: NUMBER HELP: Line delimiter ↩
    to use when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE32.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC ↩
    checksum to the base encoded data [FALSE] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE32.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR ↩
    encoding and defines corresponding parameter #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE32.ARMOR.HEADER="" # TYPE: STRING HELP: String ↩
    for the header [AUTO] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE32.ARMOR.VERSION="" # TYPE: STRING HELP: Version ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE32.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE16="" # TYPE: OBJECT HELP: Base16 encoding ( ↩
    RFC4648) #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE16.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE16.MODE="" # TYPE: NUMBER HELP: Encoding mode [ ↩
    BLOCK] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE16.LINE="" # TYPE: NUMBER HELP: Maximum length of ↩
     an encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE16.DELIM="" # TYPE: NUMBER HELP: Line delimiter ↩
    to use when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE16.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC ↩
    checksum to the base encoded data [FALSE] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE16.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR ↩
    encoding and defines corresponding parameter #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE16.ARMOR.HEADER="" # TYPE: STRING HELP: String ↩
    for the header [AUTO] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE16.ARMOR.VERSION="" # TYPE: STRING HELP: Version ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.BASE16.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.OPGP="" # TYPE: OBJECT HELP: Base64 encoding with 76 ↩
    character, CRC checksum and Armor clasp (RFC4880) #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.OPGP.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
LIM.flcl.CONV.WRITE.CHARACTER.ENCODE.OPGP.DELIM="" # TYPE: NUMBER HELP: Line delimiter to ↩
    use when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.CHARACTER.FLAM4NDC="" # TYPE: SWITCH HELP: Write data as FLAM4FILE in ↩
    MODE=NDC(No Data Compression) [FALSE] #
LIM.flcl.CONV.WRITE.CHARACTER.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character ↩
    handling [DETACH] #
LIM.flcl.CONV.WRITE.CHARACTER.FRCREC="" # TYPE: SWITCH HELP: Enforce record orientation on ↩
```

```
      record oriented devices [FALSE] #
 LIM.flcl.CONV.WRITE.CHARACTER.LANG="" # TYPE: STRING HELP: Redefine environment variable ↩
     LANG for write operation #
 LIM.flcl.CONV.WRITE.CHARACTER.PLATFORM="" # TYPE: STRING HELP: Redefine environment ↩
     variable FL_PLATFORM for write operation #
 LIM.flcl.CONV.WRITE.CHARACTER.HASH="" # TYPE: OBJECT HELP: Generation or check of one way ↩
     hash values of output data #
 LIM.flcl.CONV.WRITE.CHARACTER.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.CONV.WRITE.CHARACTER.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum ↩
     verification #
 LIM.flcl.CONV.WRITE.CHARACTER.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex- ↩
     string #
 LIM.flcl.CONV.WRITE.CHARACTER.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains ↩
     Checksum as hex-string #
 LIM.flcl.CONV.WRITE.CHARACTER.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [ ↩
     GNU] #
 LIM.flcl.CONV.WRITE.CHARACTER.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file ↩
     [LOG] #
 LIM.flcl.CONV.WRITE.TEXT="" # TYPE: OBJECT HELP: Write text data to a file #

# DESCRIPTION for LIM.flcl.CONV.WRITE.TEXT.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.CONV.WRITE.TEXT.NET="" # TYPE: OVERLAY HELP: Network parameter for remote write ↩
     access #

# DESCRIPTION for LIM.flcl.CONV.WRITE.TEXT.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.TEXT.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.CONV.WRITE.TEXT.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.WRITE.TEXT.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.WRITE.TEXT.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.TEXT.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.WRITE.TEXT.NET.IPS:
```

```
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.TEXT.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.CONV.WRITE.TEXT.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.WRITE.TEXT.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.WRITE.TEXT.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.TEXT.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.WRITE.TEXT.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.TEXT.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.CONV.WRITE.TEXT.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""]  ↩
     #
 LIM.flcl.CONV.WRITE.TEXT.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.CONV.WRITE.TEXT.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.TEXT.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.CONV.WRITE.TEXT.FILE="" # TYPE: STRING HELP: Name/URL of file to write [''== ↩
     origin.ext] #
 LIM.flcl.CONV.WRITE.TEXT.FALLOC="" # TYPE: OBJECT HELP: Platform dependent file allocation ↩
      parameter #
 LIM.flcl.CONV.WRITE.TEXT.FALLOC.ORGANIZATION="" # TYPE: NUMBER HELP: File organization [ ↩
     SEQ] #
 LIM.flcl.CONV.WRITE.TEXT.FALLOC.BLKSIZE="" # TYPE: NUMBER HELP: Block size [system] #
 LIM.flcl.CONV.WRITE.TEXT.FALLOC.RECFORMAT="" # TYPE: NUMBER HELP: Record format [auto] #
 LIM.flcl.CONV.WRITE.TEXT.FALLOC.RECLENGTH="" # TYPE: NUMBER HELP: Record length (data only ↩
     , no print control and length fields) #
 LIM.flcl.CONV.WRITE.TEXT.FALLOC.KEYDISP="" # TYPE: NUMBER HELP: Key disposition for index/ ↩
```

```
    key sequential data sets [OLD] #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.KEYPOSITION="" # TYPE: NUMBER HELP: Key position (>=1) for ↩
     index/key sequential data sets [1] #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.KEYLENGTH="" # TYPE: NUMBER HELP: Key length (>=1) for  ↩
    index/key sequential data sets [8] #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO  ↩
    subsystem (only for z/OS) #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the  ↩
    subsystem #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SUBSYSTEM.PARM="'" # TYPE: STRING HELP: Parameter list  ↩
    for the subsystem (comma separated) #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE="" # TYPE: OBJECT HELP: Platform dependent space  ↩
    definition parameter #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.PRIMARY="" # TYPE: NUMBER HELP: Primary size in  ↩
    megabytes #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.SECONDARY="" # TYPE: NUMBER HELP: Secondary size in  ↩
    megabytes #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.DIRBLOCKS="" # TYPE: NUMBER HELP: Directory blocks  ↩
    for partitioned data sets #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.VOLSER="" # TYPE: STRING HELP: Volume serial number  ↩
    #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.VOLUNIT="" # TYPE: STRING HELP: Volume unit #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.DATACLASS="" # TYPE: STRING HELP: Data class #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.MGMTCLASS="" # TYPE: STRING HELP: Management class #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.STORCLASS="" # TYPE: STRING HELP: Storage class #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.NORMALDISP="" # TYPE: NUMBER HELP: Close disposition ↩
     for normal end [CATALOG] #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.ABORTDISP="" # TYPE: NUMBER HELP: Close disposition  ↩
    for abnormal end [DELETE] #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.RELEASE="" # TYPE: NUMBER HELP: Release unused space ↩
     when file is closed [SEQ=YES, PDS=NO] #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.UNMOVABLE="" # TYPE: SWITCH HELP: Allocate space  ↩
    unmovable [FALSE] #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.CONTIG="" # TYPE: SWITCH HELP: Allocate space  ↩
    contiguously [FALSE] #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.PERMANENT="" # TYPE: SWITCH HELP: Allocate space  ↩
    permanently [FALSE] #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.SPACE.RLS="" # TYPE: NUMBER HELP: Specifies the type of  ↩
    record level sharing (RLS) being done for a specific data set [NONE] #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.LIKE="" # TYPE: STRING HELP: Specifies the name of the  ↩
    model data set from which the attributes are to be copied #
LIM.flcl.CONV.WRITE.TEXT.FALLOC.RENEW="" # TYPE: SWITCH HELP: Enforce new allocation by a  ↩
    remove of an existing file #
LIM.flcl.CONV.WRITE.TEXT.REMAIN="" # TYPE: SWITCH HELP: Remain existing files (don't  ↩
    overwrite) [FALSE] #
LIM.flcl.CONV.WRITE.TEXT.APPEND="" # TYPE: SWITCH HELP: Append data to this file [FALSE] #
LIM.flcl.CONV.WRITE.TEXT.FLUSH="" # TYPE: SWITCH HELP: Enforce flush in front of close  ↩
    file [FALSE] #
LIM.flcl.CONV.WRITE.TEXT.METHOD="" # TYPE: NUMBER HELP: Method for text formatting [SYSTEM ↩
    ] #
LIM.flcl.CONV.WRITE.TEXT.RPLTAB="" # TYPE: NUMBER HELP: Replace horizontal tabulators by  ↩
    spaces using this tab width [0 - no replacement] #
LIM.flcl.CONV.WRITE.TEXT.RPLVTB="" # TYPE: NUMBER HELP: Replace vertical tabulators by new ↩
     lines using this tab width [0 = no replacement] #
LIM.flcl.CONV.WRITE.TEXT.RPLCTR="" # TYPE: NUMBER HELP: Replace remaining control  ↩
    characters [NONE] #
LIM.flcl.CONV.WRITE.TEXT.SUPCTR="" # TYPE: SWITCH HELP: Suppress control characters [FALSE ↩
    ] (RPLCTR=DELETE) #
LIM.flcl.CONV.WRITE.TEXT.SUPTWS="" # TYPE: SWITCH HELP: Suppress trailing whitespaces [ ↩
    FALSE] #
LIM.flcl.CONV.WRITE.TEXT.CCSID="" # TYPE: STRING HELP: Conversion to this CCSID [DEFAULT]  ↩
    #
```

```
LIM.flcl.CONV.WRITE.TEXT.CASE="" # TYPE: NUMBER HELP: Define upper, lower or special case conversion [NONE] #
LIM.flcl.CONV.WRITE.TEXT.BOM="" # TYPE: SWITCH HELP: Write byte order mark in front of the output text [FALSE] #
LIM.flcl.CONV.WRITE.TEXT.ELF2NL="" # TYPE: SWITCH HELP: Replace EBCDIC line feed (0x25) by new line (0x15) on output #
LIM.flcl.CONV.WRITE.TEXT.CHRMODE="" # TYPE: NUMBER HELP: Define handling of non- convertible characters [STOP] #
LIM.flcl.CONV.WRITE.TEXT.SUBCHAR="" # TYPE: NUMBER HELP: Default substitution character list [0x1A] (maximal 8 code points) #
LIM.flcl.CONV.WRITE.TEXT.SYSTABLE="" # TYPE: NUMBER HELP: Define a preloaded system substitution table #
LIM.flcl.CONV.WRITE.TEXT.USRTABLE="" # TYPE: STRING HELP: Name of the file containing the user conversion table #
LIM.flcl.CONV.WRITE.TEXT.REPORTFILE="" # TYPE: STRING HELP: Name of the report file to log substitutions #
LIM.flcl.CONV.WRITE.TEXT.COMPRESS="" # TYPE: OVERLAY HELP: Compression procedure [NONE] #
LIM.flcl.CONV.WRITE.TEXT.COMPRESS.GZIP="" # TYPE: OBJECT HELP: Compress data compliant to RCF1952 (GZIP) #
LIM.flcl.CONV.WRITE.TEXT.COMPRESS.GZIP.LEVEL="" # TYPE: NUMBER HELP: Compression level [ AUTO==6] #
LIM.flcl.CONV.WRITE.TEXT.COMPRESS.GZIP.COMMENT="" # TYPE: STRING HELP: Comment for header [NONE] #
LIM.flcl.CONV.WRITE.TEXT.COMPRESS.GZIP.CHECK="" # TYPE: SWITCH HELP: Calculate checksum over GZIP header [FALSE] #
LIM.flcl.CONV.WRITE.TEXT.COMPRESS.BZIP2="" # TYPE: OBJECT HELP: BZIP2 data compression technique #
LIM.flcl.CONV.WRITE.TEXT.COMPRESS.BZIP2.LEVEL="" # TYPE: NUMBER HELP: Compression level ( defines block size) [AUTO] #
LIM.flcl.CONV.WRITE.TEXT.COMPRESS.XZ="" # TYPE: OBJECT HELP: XZ (LZMA) data compression technique #
LIM.flcl.CONV.WRITE.TEXT.COMPRESS.XZ.LEVEL="" # TYPE: NUMBER HELP: Compression level [AUTO ==6] #
LIM.flcl.CONV.WRITE.TEXT.COMPRESS.FLAM4="" # TYPE: OBJECT HELP: Compress data with FLAM #
LIM.flcl.CONV.WRITE.TEXT.COMPRESS.FLAM4.MODE="" # TYPE: NUMBER HELP: Compression mode [ADC ] #
LIM.flcl.CONV.WRITE.TEXT.COMPRESS.FLAM4.COMMENT="" # TYPE: STRING HELP: Comment for the member in the FLAMFILE [internal use] #
LIM.flcl.CONV.WRITE.TEXT.ENCRYPT="" # TYPE: OVERLAY HELP: Encryption procedure [NONE] #
LIM.flcl.CONV.WRITE.TEXT.ENCRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password encryption #
LIM.flcl.CONV.WRITE.TEXT.ENCRYPT.F4PWD.MODE="" # TYPE: NUMBER HELP: Encryption mode [AES] #
LIM.flcl.CONV.WRITE.TEXT.ENCRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to encrypt the FLAMFILE ['********'] #
LIM.flcl.CONV.WRITE.TEXT.ENCRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management extension (FKME) #
LIM.flcl.CONV.WRITE.TEXT.ENCRYPT.F4KME.MODE="" # TYPE: NUMBER HELP: Encryption mode [AES] #
LIM.flcl.CONV.WRITE.TEXT.ENCRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for FKME ['libfkme'] #
LIM.flcl.CONV.WRITE.TEXT.ENCRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of FKME ['FKMESTD0'] #
LIM.flcl.CONV.WRITE.TEXT.ENCRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for the call of FKME [''] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE="" # TYPE: OVERLAY HELP: Encoding procedure [NONE] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE64="" # TYPE: OBJECT HELP: Base64 encoding (RFC4648) #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE64.CHRSET="" # TYPE: NUMBER HELP: Character set [ SYSTEM] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE64.MODE="" # TYPE: NUMBER HELP: Encoding mode [BLOCK] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE64.LINE="" # TYPE: NUMBER HELP: Maximum length of an
```

```
    encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE64.DELIM="" # TYPE: NUMBER HELP: Line delimiter to use ↩
     when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE64.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC checksum  ↩
    to the base encoded data [FALSE] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE64.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR  ↩
    encoding and defines corresponding parameter #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE64.ARMOR.HEADER="" # TYPE: STRING HELP: String for the ↩
     header [AUTO] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE64.ARMOR.VERSION="" # TYPE: STRING HELP: Version  ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE64.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment  ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE32="" # TYPE: OBJECT HELP: Base32 encoding (RFC4648) #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE32.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE32.MODE="" # TYPE: NUMBER HELP: Encoding mode [BLOCK]  ↩
    #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE32.LINE="" # TYPE: NUMBER HELP: Maximum length of an  ↩
    encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE32.DELIM="" # TYPE: NUMBER HELP: Line delimiter to use ↩
     when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE32.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC checksum  ↩
    to the base encoded data [FALSE] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE32.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR  ↩
    encoding and defines corresponding parameter #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE32.ARMOR.HEADER="" # TYPE: STRING HELP: String for the ↩
     header [AUTO] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE32.ARMOR.VERSION="" # TYPE: STRING HELP: Version  ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE32.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment  ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE16="" # TYPE: OBJECT HELP: Base16 encoding (RFC4648) #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE16.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE16.MODE="" # TYPE: NUMBER HELP: Encoding mode [BLOCK]  ↩
    #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE16.LINE="" # TYPE: NUMBER HELP: Maximum length of an  ↩
    encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE16.DELIM="" # TYPE: NUMBER HELP: Line delimiter to use ↩
     when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE16.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC checksum  ↩
    to the base encoded data [FALSE] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE16.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR  ↩
    encoding and defines corresponding parameter #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE16.ARMOR.HEADER="" # TYPE: STRING HELP: String for the ↩
     header [AUTO] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE16.ARMOR.VERSION="" # TYPE: STRING HELP: Version  ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.BASE16.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment  ↩
    string as key value [AUTO] #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.OPGP="" # TYPE: OBJECT HELP: Base64 encoding with 76  ↩
    character, CRC checksum and Armor clasp (RFC4880) #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.OPGP.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM] ↩
     #
LIM.flcl.CONV.WRITE.TEXT.ENCODE.OPGP.DELIM="" # TYPE: NUMBER HELP: Line delimiter to use  ↩
    when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.TEXT.FLAM4NDC="" # TYPE: SWITCH HELP: Write data as FLAM4FILE in MODE= ↩
    NDC(No Data Compression) [FALSE] #
LIM.flcl.CONV.WRITE.TEXT.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character  ↩
    handling [DETACH] #
LIM.flcl.CONV.WRITE.TEXT.FRCREC="" # TYPE: SWITCH HELP: Enforce record orientation on  ↩
```

```
    record oriented devices [FALSE] #
 LIM.flcl.CONV.WRITE.TEXT.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG  ↩
    for write operation #
 LIM.flcl.CONV.WRITE.TEXT.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
    FL_PLATFORM for write operation #
 LIM.flcl.CONV.WRITE.TEXT.DUMP="" # TYPE: SWITCH HELP: Write text dump of binary data [ ↩
    FALSE] #
 LIM.flcl.CONV.WRITE.TEXT.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash  ↩
    values of output data #
 LIM.flcl.CONV.WRITE.TEXT.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.CONV.WRITE.TEXT.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
 LIM.flcl.CONV.WRITE.TEXT.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string #
 LIM.flcl.CONV.WRITE.TEXT.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
 LIM.flcl.CONV.WRITE.TEXT.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU]  ↩
    #
 LIM.flcl.CONV.WRITE.TEXT.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG] ↩
     #
 LIM.flcl.CONV.WRITE.TEXT.INDSIZ="-1" # TYPE: NUMBER HELP: Number of XML indentation  ↩
    characters per indentation level [2 (SPACE) or 1 (TAB)] #
 LIM.flcl.CONV.WRITE.TEXT.INDCHR="" # TYPE: NUMBER HELP: Indentation character used to  ↩
    indent XML tags [SPACE] #
 LIM.flcl.CONV.WRITE.TEXT.NOCMNT="" # TYPE: SWITCH HELP: Do not write out XML comments [ ↩
    FALSE] #
 LIM.flcl.CONV.WRITE.XML="" # TYPE: OBJECT HELP: Write XML data to a file #

# DESCRIPTION for LIM.flcl.CONV.WRITE.XML.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.CONV.WRITE.XML.NET="" # TYPE: OVERLAY HELP: Network parameter for remote write  ↩
    access #

# DESCRIPTION for LIM.flcl.CONV.WRITE.XML.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.XML.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #
```

```
 LIM.flcl.CONV.WRITE.XML.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.WRITE.XML.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.WRITE.XML.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.XML.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

# DESCRIPTION for LIM.flcl.CONV.WRITE.XML.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.XML.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.CONV.WRITE.XML.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.WRITE.XML.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.WRITE.XML.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.XML.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

# DESCRIPTION for LIM.flcl.CONV.WRITE.XML.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.XML.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.CONV.WRITE.XML.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
 LIM.flcl.CONV.WRITE.XML.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.CONV.WRITE.XML.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.XML.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

 LIM.flcl.CONV.WRITE.XML.FILE="" # TYPE: STRING HELP: Name/URL of file to write [''==origin ↩
     .ext] #
 LIM.flcl.CONV.WRITE.XML.FALLOC="" # TYPE: OBJECT HELP: Platform dependent file allocation  ↩
     parameter #
 LIM.flcl.CONV.WRITE.XML.FALLOC.ORGANIZATION="" # TYPE: NUMBER HELP: File organization [SEQ ↩
```

```
    ] #
LIM.flcl.CONV.WRITE.XML.FALLOC.BLKSIZE="" # TYPE: NUMBER HELP: Block size [system] #
LIM.flcl.CONV.WRITE.XML.FALLOC.RECFORMAT="" # TYPE: NUMBER HELP: Record format [auto] #
LIM.flcl.CONV.WRITE.XML.FALLOC.RECLENGTH="" # TYPE: NUMBER HELP: Record length (data only, ↩
     no print control and length fields) #
LIM.flcl.CONV.WRITE.XML.FALLOC.KEYDISP="" # TYPE: NUMBER HELP: Key disposition for index/ ↩
    key sequential data sets [OLD] #
LIM.flcl.CONV.WRITE.XML.FALLOC.KEYPOSITION="" # TYPE: NUMBER HELP: Key position (>=1) for  ↩
    index/key sequential data sets [1] #
LIM.flcl.CONV.WRITE.XML.FALLOC.KEYLENGTH="" # TYPE: NUMBER HELP: Key length (>=1) for  ↩
    index/key sequential data sets [8] #
LIM.flcl.CONV.WRITE.XML.FALLOC.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO  ↩
    subsystem (only for z/OS) #
LIM.flcl.CONV.WRITE.XML.FALLOC.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the  ↩
    subsystem #
LIM.flcl.CONV.WRITE.XML.FALLOC.SUBSYSTEM.PARM="'" # TYPE: STRING HELP: Parameter list for ↩
     the subsystem (comma separated) #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE="" # TYPE: OBJECT HELP: Platform dependent space  ↩
    definition parameter #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.PRIMARY="" # TYPE: NUMBER HELP: Primary size in  ↩
    megabytes #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.SECONDARY="" # TYPE: NUMBER HELP: Secondary size in  ↩
    megabytes #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.DIRBLOCKS="" # TYPE: NUMBER HELP: Directory blocks  ↩
    for partitioned data sets #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.VOLSER="" # TYPE: STRING HELP: Volume serial number #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.VOLUNIT="" # TYPE: STRING HELP: Volume unit #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.DATACLASS="" # TYPE: STRING HELP: Data class #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.MGMTCLASS="" # TYPE: STRING HELP: Management class #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.STORCLASS="" # TYPE: STRING HELP: Storage class #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.NORMALDISP="" # TYPE: NUMBER HELP: Close disposition  ↩
    for normal end [CATALOG] #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.ABORTDISP="" # TYPE: NUMBER HELP: Close disposition  ↩
    for abnormal end [DELETE] #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.RELEASE="" # TYPE: NUMBER HELP: Release unused space  ↩
    when file is closed [SEQ=YES, PDS=NO] #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.UNMOVABLE="" # TYPE: SWITCH HELP: Allocate space  ↩
    unmovable [FALSE] #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.CONTIG="" # TYPE: SWITCH HELP: Allocate space  ↩
    contiguously [FALSE] #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.PERMANENT="" # TYPE: SWITCH HELP: Allocate space  ↩
    permanently [FALSE] #
LIM.flcl.CONV.WRITE.XML.FALLOC.SPACE.RLS="" # TYPE: NUMBER HELP: Specifies the type of  ↩
    record level sharing (RLS) being done for a specific data set [NONE] #
LIM.flcl.CONV.WRITE.XML.FALLOC.LIKE="" # TYPE: STRING HELP: Specifies the name of the  ↩
    model data set from which the attributes are to be copied #
LIM.flcl.CONV.WRITE.XML.FALLOC.RENEW="" # TYPE: SWITCH HELP: Enforce new allocation by a  ↩
    remove of an existing file #
LIM.flcl.CONV.WRITE.XML.REMAIN="" # TYPE: SWITCH HELP: Remain existing files (don't  ↩
    overwrite) [FALSE] #
LIM.flcl.CONV.WRITE.XML.APPEND="" # TYPE: SWITCH HELP: Append data to this file [FALSE] #
LIM.flcl.CONV.WRITE.XML.FLUSH="" # TYPE: SWITCH HELP: Enforce flush in front of close file ↩
     [FALSE] #
LIM.flcl.CONV.WRITE.XML.METHOD="" # TYPE: OVERLAY HELP: Method for XML formatting [ ↩
    STANDARD] #
LIM.flcl.CONV.WRITE.XML.METHOD.STANDARD="" # TYPE: OBJECT HELP: Reproduce all XML elements ↩
     as is #
LIM.flcl.CONV.WRITE.XML.METHOD.STANDARD.NOCMNT="" # TYPE: SWITCH HELP: Do not write out  ↩
    XML comments [FALSE] #
LIM.flcl.CONV.WRITE.XML.METHOD.STANDARD.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size ↩
     for preallocation [65536] #
LIM.flcl.CONV.WRITE.XML.METHOD.STANDARD.INICNT="" # TYPE: NUMBER HELP: Initial amount of  ↩
```

```
    records for preallocation [128] #
LIM.flcl.CONV.WRITE.XML.METHOD.PRETTYPRINT="" # TYPE: OBJECT HELP: Indentation of XML tags ↩
     for human readability #
LIM.flcl.CONV.WRITE.XML.METHOD.PRETTYPRINT.MAXWSP="" # TYPE: NUMBER HELP: Size of internal ↩
     whitespace cache (only used when method is minimize or pretty printing) [4096] #
LIM.flcl.CONV.WRITE.XML.METHOD.PRETTYPRINT.WSPERR="" # TYPE: NUMBER HELP: Action when the  ↩
    whitespace cache overflows [WRITEOUT] #
LIM.flcl.CONV.WRITE.XML.METHOD.PRETTYPRINT.INDSIZ="-1" # TYPE: NUMBER HELP: Number of  ↩
    indentation characters per indentation level [2 (SPACE) or 1 (TAB)] #
LIM.flcl.CONV.WRITE.XML.METHOD.PRETTYPRINT.INDCHR="" # TYPE: NUMBER HELP: Indentation  ↩
    character used to indent XML tags [SPACE] #
LIM.flcl.CONV.WRITE.XML.METHOD.PRETTYPRINT.NOCMNT="" # TYPE: SWITCH HELP: Do not write out ↩
     XML comments [FALSE] #
LIM.flcl.CONV.WRITE.XML.METHOD.PRETTYPRINT.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer  ↩
    size for preallocation [65536] #
LIM.flcl.CONV.WRITE.XML.METHOD.PRETTYPRINT.INICNT="" # TYPE: NUMBER HELP: Initial amount  ↩
    of elements for preallocation [128] #
LIM.flcl.CONV.WRITE.XML.METHOD.MINIMIZED="" # TYPE: OBJECT HELP: Discards comments,  ↩
    insignificant whitespace and whitespace-only sections between tags #
LIM.flcl.CONV.WRITE.XML.METHOD.MINIMIZED.MAXWSP="" # TYPE: NUMBER HELP: Size of internal  ↩
    whitespace cache (only used when method is minimize or pretty printing) [4096] #
LIM.flcl.CONV.WRITE.XML.METHOD.MINIMIZED.WSPERR="" # TYPE: NUMBER HELP: Action when the  ↩
    whitespace cache overflows [WRITEOUT] #
LIM.flcl.CONV.WRITE.XML.METHOD.MINIMIZED.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer  ↩
    size for preallocation [65536] #
LIM.flcl.CONV.WRITE.XML.METHOD.MINIMIZED.INICNT="" # TYPE: NUMBER HELP: Initial amount of  ↩
    elements for preallocation [128] #
LIM.flcl.CONV.WRITE.XML.CCSID="" # TYPE: STRING HELP: Conversion to this CCSID [no  ↩
    character conversion, leave in UTF-8] #
LIM.flcl.CONV.WRITE.XML.BOM="" # TYPE: SWITCH HELP: Write byte order mark in front of the  ↩
    output text [FALSE] #
LIM.flcl.CONV.WRITE.XML.ELF2NL="" # TYPE: SWITCH HELP: Replace EBCDIC line feed (0x25) by  ↩
    new line (0x15) on output #
LIM.flcl.CONV.WRITE.XML.CHRMODE="" # TYPE: NUMBER HELP: Define handling of non-convertible ↩
     characters [STOP] #
LIM.flcl.CONV.WRITE.XML.SUBCHAR="" # TYPE: NUMBER HELP: Default substitution character  ↩
    list [0x1A] (maximal 8 code points) #
LIM.flcl.CONV.WRITE.XML.SYSTABLE="" # TYPE: NUMBER HELP: Define a preloaded system  ↩
    substitution table #
LIM.flcl.CONV.WRITE.XML.USRTABLE="" # TYPE: STRING HELP: Name of the file containing the  ↩
    user conversion table #
LIM.flcl.CONV.WRITE.XML.REPORTFILE="" # TYPE: STRING HELP: Name of the report file to log  ↩
    substitutions #
LIM.flcl.CONV.WRITE.XML.COMPRESS="" # TYPE: OVERLAY HELP: Compression procedure [NONE] #
LIM.flcl.CONV.WRITE.XML.COMPRESS.GZIP="" # TYPE: OBJECT HELP: Compress data compliant to  ↩
    RCF1952 (GZIP) #
LIM.flcl.CONV.WRITE.XML.COMPRESS.GZIP.LEVEL="" # TYPE: NUMBER HELP: Compression level [ ↩
    AUTO==6] #
LIM.flcl.CONV.WRITE.XML.COMPRESS.GZIP.COMMENT="" # TYPE: STRING HELP: Comment for header [ ↩
    NONE] #
LIM.flcl.CONV.WRITE.XML.COMPRESS.GZIP.CHECK="" # TYPE: SWITCH HELP: Calculate checksum  ↩
    over GZIP header [FALSE] #
LIM.flcl.CONV.WRITE.XML.COMPRESS.BZIP2="" # TYPE: OBJECT HELP: BZIP2 data compression  ↩
    technique #
LIM.flcl.CONV.WRITE.XML.COMPRESS.BZIP2.LEVEL="" # TYPE: NUMBER HELP: Compression level ( ↩
    defines block size) [AUTO] #
LIM.flcl.CONV.WRITE.XML.COMPRESS.XZ="" # TYPE: OBJECT HELP: XZ (LZMA) data compression  ↩
    technique #
LIM.flcl.CONV.WRITE.XML.COMPRESS.XZ.LEVEL="" # TYPE: NUMBER HELP: Compression level [AUTO ↩
    ==6] #
LIM.flcl.CONV.WRITE.XML.COMPRESS.FLAM4="" # TYPE: OBJECT HELP: Compress data with FLAM #
LIM.flcl.CONV.WRITE.XML.COMPRESS.FLAM4.MODE="" # TYPE: NUMBER HELP: Compression mode [ADC] ↩
```

```
     #
LIM.flcl.CONV.WRITE.XML.COMPRESS.FLAM4.COMMENT="" # TYPE: STRING HELP: Comment for the ↩
    member in the FLAMFILE [internal use] #
LIM.flcl.CONV.WRITE.XML.ENCRYPT="" # TYPE: OVERLAY HELP: Encryption procedure [NONE] #
LIM.flcl.CONV.WRITE.XML.ENCRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password encryption ↩
     #
LIM.flcl.CONV.WRITE.XML.ENCRYPT.F4PWD.MODE="" # TYPE: NUMBER HELP: Encryption mode [AES] #
LIM.flcl.CONV.WRITE.XML.ENCRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to ↩
    encrypt the FLAMFILE ['********'] #
LIM.flcl.CONV.WRITE.XML.ENCRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management ↩
    extension (FKME) #
LIM.flcl.CONV.WRITE.XML.ENCRYPT.F4KME.MODE="" # TYPE: NUMBER HELP: Encryption mode [AES] #
LIM.flcl.CONV.WRITE.XML.ENCRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for ↩
    FKME ['libfkme'] #
LIM.flcl.CONV.WRITE.XML.ENCRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of ↩
    FKME ['FKMESTD0'] #
LIM.flcl.CONV.WRITE.XML.ENCRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for the ↩
    call of FKME [''] #
LIM.flcl.CONV.WRITE.XML.ENCODE="" # TYPE: OVERLAY HELP: Encoding procedure [NONE] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE64="" # TYPE: OBJECT HELP: Base64 encoding (RFC4648) #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE64.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM ↩
    ] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE64.MODE="" # TYPE: NUMBER HELP: Encoding mode [BLOCK] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE64.LINE="" # TYPE: NUMBER HELP: Maximum length of an ↩
    encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE64.DELIM="" # TYPE: NUMBER HELP: Line delimiter to use ↩
    when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE64.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC checksum to ↩
     the base encoded data [FALSE] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE64.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR ↩
    encoding and defines corresponding parameter #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE64.ARMOR.HEADER="" # TYPE: STRING HELP: String for the ↩
    header [AUTO] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE64.ARMOR.VERSION="" # TYPE: STRING HELP: Version string ↩
     as key value [AUTO] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE64.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment string ↩
     as key value [AUTO] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE32="" # TYPE: OBJECT HELP: Base32 encoding (RFC4648) #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE32.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM ↩
    ] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE32.MODE="" # TYPE: NUMBER HELP: Encoding mode [BLOCK] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE32.LINE="" # TYPE: NUMBER HELP: Maximum length of an ↩
    encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE32.DELIM="" # TYPE: NUMBER HELP: Line delimiter to use ↩
    when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE32.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC checksum to ↩
     the base encoded data [FALSE] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE32.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR ↩
    encoding and defines corresponding parameter #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE32.ARMOR.HEADER="" # TYPE: STRING HELP: String for the ↩
    header [AUTO] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE32.ARMOR.VERSION="" # TYPE: STRING HELP: Version string ↩
     as key value [AUTO] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE32.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment string ↩
     as key value [AUTO] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE16="" # TYPE: OBJECT HELP: Base16 encoding (RFC4648) #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE16.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM ↩
    ] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE16.MODE="" # TYPE: NUMBER HELP: Encoding mode [BLOCK] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE16.LINE="" # TYPE: NUMBER HELP: Maximum length of an ↩
    encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.XML.ENCODE.BASE16.DELIM="" # TYPE: NUMBER HELP: Line delimiter to use ↩
```

```
    when mode is BLOCK and line>0 [SYSTEM] #
 LIM.flcl.CONV.WRITE.XML.ENCODE.BASE16.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC checksum to ↩
     the base encoded data [FALSE] #
 LIM.flcl.CONV.WRITE.XML.ENCODE.BASE16.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR  ↩
    encoding and defines corresponding parameter #
 LIM.flcl.CONV.WRITE.XML.ENCODE.BASE16.ARMOR.HEADER="" # TYPE: STRING HELP: String for the  ↩
    header [AUTO] #
 LIM.flcl.CONV.WRITE.XML.ENCODE.BASE16.ARMOR.VERSION="" # TYPE: STRING HELP: Version string ↩
     as key value [AUTO] #
 LIM.flcl.CONV.WRITE.XML.ENCODE.BASE16.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment string ↩
     as key value [AUTO] #
 LIM.flcl.CONV.WRITE.XML.ENCODE.OPGP="" # TYPE: OBJECT HELP: Base64 encoding with 76  ↩
    character, CRC checksum and Armor clasp (RFC4880) #
 LIM.flcl.CONV.WRITE.XML.ENCODE.OPGP.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM]  ↩
    #
 LIM.flcl.CONV.WRITE.XML.ENCODE.OPGP.DELIM="" # TYPE: NUMBER HELP: Line delimiter to use  ↩
    when mode is BLOCK and line>0 [SYSTEM] #
 LIM.flcl.CONV.WRITE.XML.FLAM4NDC="" # TYPE: SWITCH HELP: Write data as FLAM4FILE in MODE= ↩
    NDC(No Data Compression) [FALSE] #
 LIM.flcl.CONV.WRITE.XML.FRCREC="" # TYPE: SWITCH HELP: Enforce record orientation on  ↩
    record oriented devices [FALSE] #
 LIM.flcl.CONV.WRITE.XML.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG  ↩
    for write operation #
 LIM.flcl.CONV.WRITE.XML.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
    FL_PLATFORM for write operation #
 LIM.flcl.CONV.WRITE.XML.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash  ↩
    values of output data #
 LIM.flcl.CONV.WRITE.XML.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.CONV.WRITE.XML.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
 LIM.flcl.CONV.WRITE.XML.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string #
 LIM.flcl.CONV.WRITE.XML.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
 LIM.flcl.CONV.WRITE.XML.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] #
 LIM.flcl.CONV.WRITE.XML.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG]  ↩
    #
 LIM.flcl.CONV.WRITE.RECORD="" # TYPE: OBJECT HELP: Write records to a file #

# DESCRIPTION for LIM.flcl.CONV.WRITE.RECORD.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.CONV.WRITE.RECORD.NET="" # TYPE: OVERLAY HELP: Network parameter for remote write ↩
     access #

# DESCRIPTION for LIM.flcl.CONV.WRITE.RECORD.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
```

```
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.RECORD.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.CONV.WRITE.RECORD.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.WRITE.RECORD.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.WRITE.RECORD.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.RECORD.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.WRITE.RECORD.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.RECORD.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.CONV.WRITE.RECORD.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.WRITE.RECORD.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.WRITE.RECORD.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.RECORD.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.WRITE.RECORD.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.RECORD.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #
```

```
LIM.flcl.CONV.WRITE.RECORD.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name ↩
    [""] #
LIM.flcl.CONV.WRITE.RECORD.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
LIM.flcl.CONV.WRITE.RECORD.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
LIM.flcl.CONV.WRITE.RECORD.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
    optional] #

LIM.flcl.CONV.WRITE.RECORD.FILE="" # TYPE: STRING HELP: Name/URL of file to write [''== ↩
    origin.ext] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC="" # TYPE: OBJECT HELP: Platform dependent file ↩
    allocation parameter #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.ORGANIZATION="" # TYPE: NUMBER HELP: File organization [ ↩
    SEQ] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.BLKSIZE="" # TYPE: NUMBER HELP: Block size [system] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.RECFORMAT="" # TYPE: NUMBER HELP: Record format [auto] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.RECLENGTH="" # TYPE: NUMBER HELP: Record length (data ↩
    only, no print control and length fields) #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.KEYDISP="" # TYPE: NUMBER HELP: Key disposition for ↩
    index/key sequential data sets [OLD] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.KEYPOSITION="" # TYPE: NUMBER HELP: Key position (>=1) ↩
    for index/key sequential data sets [1] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.KEYLENGTH="" # TYPE: NUMBER HELP: Key length (>=1) for ↩
    index/key sequential data sets [8] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO ↩
    subsystem (only for z/OS) #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the ↩
     subsystem #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list ↩
    for the subsystem (comma separated) #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE="" # TYPE: OBJECT HELP: Platform dependent space ↩
    definition parameter #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.PRIMARY="" # TYPE: NUMBER HELP: Primary size in ↩
    megabytes #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.SECONDARY="" # TYPE: NUMBER HELP: Secondary size ↩
    in megabytes #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.DIRBLOCKS="" # TYPE: NUMBER HELP: Directory blocks ↩
     for partitioned data sets #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.VOLSER="" # TYPE: STRING HELP: Volume serial ↩
    number #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.VOLUNIT="" # TYPE: STRING HELP: Volume unit #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.DATACLASS="" # TYPE: STRING HELP: Data class #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.MGMTCLASS="" # TYPE: STRING HELP: Management class ↩
     #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.STORCLASS="" # TYPE: STRING HELP: Storage class #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.NORMALDISP="" # TYPE: NUMBER HELP: Close ↩
    disposition for normal end [CATALOG] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.ABORTDISP="" # TYPE: NUMBER HELP: Close ↩
    disposition for abnormal end [DELETE] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.RELEASE="" # TYPE: NUMBER HELP: Release unused ↩
    space when file is closed [SEQ=YES, PDS=NO] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.UNMOVABLE="" # TYPE: SWITCH HELP: Allocate space ↩
    unmovable [FALSE] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.CONTIG="" # TYPE: SWITCH HELP: Allocate space ↩
    contiguously [FALSE] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.PERMANENT="" # TYPE: SWITCH HELP: Allocate space ↩
    permanently [FALSE] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.SPACE.RLS="" # TYPE: NUMBER HELP: Specifies the type of ↩
    record level sharing (RLS) being done for a specific data set [NONE] #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.LIKE="" # TYPE: STRING HELP: Specifies the name of the ↩
    model data set from which the attributes are to be copied #
LIM.flcl.CONV.WRITE.RECORD.FALLOC.RENEW="" # TYPE: SWITCH HELP: Enforce new allocation by ↩
    a remove of an existing file #
```

```
LIM.flcl.CONV.WRITE.RECORD.REMAIN="" # TYPE: SWITCH HELP: Remain existing files (don't overwrite) [FALSE] #
LIM.flcl.CONV.WRITE.RECORD.APPEND="" # TYPE: SWITCH HELP: Append data to this file [FALSE] #
LIM.flcl.CONV.WRITE.RECORD.FLUSH="" # TYPE: SWITCH HELP: Enforce flush in front of close file [FALSE] #
LIM.flcl.CONV.WRITE.RECORD.RECMODE="" # TYPE: NUMBER HELP: Mode used to write records [STOP] #
LIM.flcl.CONV.WRITE.RECORD.RECDELIM="" # TYPE: STRING HELP: Delimiter used to define end of record #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT="" # TYPE: OVERLAY HELP: Format of the length field for open platforms [word] #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.INTEGER="" # TYPE: OBJECT HELP: Use integer format for the length field #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.INTEGER.ENDIAN="" # TYPE: NUMBER HELP: Byte order [SYSTEM] #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.INTEGER.WIDTH="" # TYPE: NUMBER HELP: Bit width/rate [B32] #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.INTEGER.INCLUDE="" # TYPE: SWITCH HELP: Include length field in length value [OFF] #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.STRING="" # TYPE: OBJECT HELP: Use string format for the length field #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.STRING.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM] #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.STRING.COUNT="" # TYPE: NUMBER HELP: Character count [C06] #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.STRING.BASE="" # TYPE: NUMBER HELP: String base [B10] #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.STRING.INCLUDE="" # TYPE: SWITCH HELP: Include length field in length value [OFF] #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.BCD="" # TYPE: OBJECT HELP: Use BCD format for the length field #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.BCD.WIDTH="" # TYPE: NUMBER HELP: Bit width/rate [B32] #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.BCD.INCLUDE="" # TYPE: SWITCH HELP: Include length field in length value [OFF] #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.HOST="" # TYPE: OBJECT HELP: Use host format (LLxx) for the length field #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.HOST.ENDIAN="" # TYPE: NUMBER HELP: Byte order [BIG ENDIAN] for first 16 Bit #
LIM.flcl.CONV.WRITE.RECORD.LENFORMAT.HOST.EXCLUDE="" # TYPE: SWITCH HELP: Record length field not part of the length value [OFF] #
LIM.flcl.CONV.WRITE.RECORD.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character handling [DETACH] #
LIM.flcl.CONV.WRITE.RECORD.SUPPADDING="" # TYPE: SWITCH HELP: Suppress trailing padding characters (useful for variable records) #
LIM.flcl.CONV.WRITE.RECORD.PADCHAR="" # TYPE: STRING HELP: Character to pad records to a fixed length [system] #
LIM.flcl.CONV.WRITE.RECORD.CCSID="" # TYPE: STRING HELP: Conversion to this CCSID [DEFAULT] #
LIM.flcl.CONV.WRITE.RECORD.CASE="" # TYPE: NUMBER HELP: Define upper, lower or special case conversion [NONE] #
LIM.flcl.CONV.WRITE.RECORD.BOM="" # TYPE: SWITCH HELP: Write byte order mark in front of each record [FALSE] #
LIM.flcl.CONV.WRITE.RECORD.CHRMODE="" # TYPE: NUMBER HELP: Define handling of non- convertible characters [STOP] #
LIM.flcl.CONV.WRITE.RECORD.SUBCHAR="" # TYPE: NUMBER HELP: Default substitution character list [0x1A] (maximal 8 code points) #
LIM.flcl.CONV.WRITE.RECORD.SYSTABLE="" # TYPE: NUMBER HELP: Define a preloaded system substitution table #
LIM.flcl.CONV.WRITE.RECORD.USRTABLE="" # TYPE: STRING HELP: Name of the file containing the user conversion table #
```

```
LIM.flcl.CONV.WRITE.RECORD.REPORTFILE="" # TYPE: STRING HELP: Name of the report file to log substitutions #
LIM.flcl.CONV.WRITE.RECORD.COMPRESS="" # TYPE: OVERLAY HELP: Compression procedure [NONE] #
LIM.flcl.CONV.WRITE.RECORD.COMPRESS.FLAM4="" # TYPE: OBJECT HELP: Compress records with FLAM #
LIM.flcl.CONV.WRITE.RECORD.COMPRESS.FLAM4.MODE="" # TYPE: NUMBER HELP: Compression mode [ADC] #
LIM.flcl.CONV.WRITE.RECORD.COMPRESS.FLAM4.COMMENT="" # TYPE: STRING HELP: Comment for the member in the FLAMFILE [internal use] #
LIM.flcl.CONV.WRITE.RECORD.ENCRYPT="" # TYPE: OVERLAY HELP: Encryption procedure [NONE] #
LIM.flcl.CONV.WRITE.RECORD.ENCRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password encryption #
LIM.flcl.CONV.WRITE.RECORD.ENCRYPT.F4PWD.MODE="" # TYPE: NUMBER HELP: Encryption mode [AES] #
LIM.flcl.CONV.WRITE.RECORD.ENCRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to encrypt the FLAMFILE ['********'] #
LIM.flcl.CONV.WRITE.RECORD.ENCRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management extension (FKME) #
LIM.flcl.CONV.WRITE.RECORD.ENCRYPT.F4KME.MODE="" # TYPE: NUMBER HELP: Encryption mode [AES] #
LIM.flcl.CONV.WRITE.RECORD.ENCRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for FKME ['libfkme'] #
LIM.flcl.CONV.WRITE.RECORD.ENCRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of FKME ['FKMESTD0'] #
LIM.flcl.CONV.WRITE.RECORD.ENCRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for the call of FKME [''] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE="" # TYPE: OVERLAY HELP: Encoding procedure [NONE] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE64="" # TYPE: OBJECT HELP: Base64 encoding (RFC4648) #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE64.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE64.MODE="" # TYPE: NUMBER HELP: Encoding mode [BLOCK] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE64.LINE="" # TYPE: NUMBER HELP: Maximum length of an encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE64.DELIM="" # TYPE: NUMBER HELP: Line delimiter to use when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE64.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC checksum to the base encoded data [FALSE] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE64.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR encoding and defines corresponding parameter #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE64.ARMOR.HEADER="" # TYPE: STRING HELP: String for the header [AUTO] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE64.ARMOR.VERSION="" # TYPE: STRING HELP: Version string as key value [AUTO] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE64.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment string as key value [AUTO] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE32="" # TYPE: OBJECT HELP: Base32 encoding (RFC4648) #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE32.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE32.MODE="" # TYPE: NUMBER HELP: Encoding mode [BLOCK] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE32.LINE="" # TYPE: NUMBER HELP: Maximum length of an encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE32.DELIM="" # TYPE: NUMBER HELP: Line delimiter to use when mode is BLOCK and line>0 [SYSTEM] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE32.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC checksum to the base encoded data [FALSE] #
LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE32.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR encoding and defines corresponding parameter #
```

```
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE32.ARMOR.HEADER="" # TYPE: STRING HELP: String for  ↩
    the header [AUTO] #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE32.ARMOR.VERSION="" # TYPE: STRING HELP: Version  ↩
    string as key value [AUTO] #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE32.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment  ↩
    string as key value [AUTO] #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE16="" # TYPE: OBJECT HELP: Base16 encoding (RFC4648) ↩
     #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE16.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE16.MODE="" # TYPE: NUMBER HELP: Encoding mode [BLOCK ↩
    ] #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE16.LINE="" # TYPE: NUMBER HELP: Maximum length of an ↩
     encoded line when mode is BLOCK [0=unlimited] #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE16.DELIM="" # TYPE: NUMBER HELP: Line delimiter to  ↩
    use when mode is BLOCK and line>0 [SYSTEM] #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE16.CRCCHK="" # TYPE: SWITCH HELP: Add a CRC checksum ↩
     to the base encoded data [FALSE] #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE16.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR  ↩
    encoding and defines corresponding parameter #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE16.ARMOR.HEADER="" # TYPE: STRING HELP: String for  ↩
    the header [AUTO] #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE16.ARMOR.VERSION="" # TYPE: STRING HELP: Version  ↩
    string as key value [AUTO] #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.BASE16.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment  ↩
    string as key value [AUTO] #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.OPGP="" # TYPE: OBJECT HELP: Base64 encoding with 76  ↩
    character, CRC checksum and Armor clasp (RFC4880) #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.OPGP.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
 LIM.flcl.CONV.WRITE.RECORD.ENCODE.OPGP.DELIM="" # TYPE: NUMBER HELP: Line delimiter to use ↩
     when mode is BLOCK and line>0 [SYSTEM] #
 LIM.flcl.CONV.WRITE.RECORD.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG ↩
     for write operation #
 LIM.flcl.CONV.WRITE.RECORD.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
    FL_PLATFORM for write operation #
 LIM.flcl.CONV.WRITE.RECORD.HASH="" # TYPE: OBJECT HELP: Generation or check of one way  ↩
    hash values of output data #
 LIM.flcl.CONV.WRITE.RECORD.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.CONV.WRITE.RECORD.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
 LIM.flcl.CONV.WRITE.RECORD.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string ↩
     #
 LIM.flcl.CONV.WRITE.RECORD.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
 LIM.flcl.CONV.WRITE.RECORD.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU ↩
    ] #
 LIM.flcl.CONV.WRITE.RECORD.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [ ↩
    LOG] #
 LIM.flcl.CONV.WRITE.RECORD.INDSIZ="-1" # TYPE: NUMBER HELP: Number of XML indentation  ↩
    characters per indentation level [2 (SPACE) or 1 (TAB)] #
 LIM.flcl.CONV.WRITE.RECORD.INDCHR="" # TYPE: NUMBER HELP: Indentation character used to  ↩
    indent XML tags [SPACE] #
 LIM.flcl.CONV.WRITE.RECORD.NOCMNT="" # TYPE: SWITCH HELP: Do not write out XML comments [ ↩
    FALSE] #
 LIM.flcl.CONV.WRITE.FLAM4="" # TYPE: OBJECT HELP: Write records to a FLAM4FILE #

# DESCRIPTION for LIM.flcl.CONV.WRITE.FLAM4.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.
```

```
This feature requires a FLIES server.


 #
 LIM.flcl.CONV.WRITE.FLAM4.NET="" # TYPE: OVERLAY HELP: Network parameter for remote write  ↩
     access #

# DESCRIPTION for LIM.flcl.CONV.WRITE.FLAM4.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.FLAM4.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.CONV.WRITE.FLAM4.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.WRITE.FLAM4.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.WRITE.FLAM4.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.FLAM4.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.WRITE.FLAM4.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.FLAM4.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.CONV.WRITE.FLAM4.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.CONV.WRITE.FLAM4.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.CONV.WRITE.FLAM4.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
```

```
 LIM.flcl.CONV.WRITE.FLAM4.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.CONV.WRITE.FLAM4.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.CONV.WRITE.FLAM4.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.CONV.WRITE.FLAM4.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] ↩
      #
 LIM.flcl.CONV.WRITE.FLAM4.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.CONV.WRITE.FLAM4.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.CONV.WRITE.FLAM4.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.CONV.WRITE.FLAM4.FILE="" # TYPE: STRING HELP: Name/URL of the FLAMFILE to write  ↩
     [''==stdout] #
 LIM.flcl.CONV.WRITE.FLAM4.REMAIN="" # TYPE: SWITCH HELP: Remain existing files (don't  ↩
     overwrite) [FALSE] #
 LIM.flcl.CONV.WRITE.FLAM4.APPEND="" # TYPE: SWITCH HELP: Append the member to this  ↩
     FLAMFILE (not useful with encryption) [FALSE] #
 LIM.flcl.CONV.WRITE.FLAM4.NOPATH="" # TYPE: SWITCH HELP: Exclude directory path of the  ↩
     original file in the compressed data [FALSE] #
 LIM.flcl.CONV.WRITE.FLAM4.SPACE="" # TYPE: OBJECT HELP: Space parameter for FLAMFILE  ↩
     allocation (mainframes only) #
 LIM.flcl.CONV.WRITE.FLAM4.SPACE.PRIMARY="" # TYPE: NUMBER HELP: Primary size in megabytes  ↩
     #
 LIM.flcl.CONV.WRITE.FLAM4.SPACE.SECONDARY="" # TYPE: NUMBER HELP: Secondary size in  ↩
     megabytes #
 LIM.flcl.CONV.WRITE.FLAM4.SPACE.VOLSER="" # TYPE: STRING HELP: Volume serial number #
 LIM.flcl.CONV.WRITE.FLAM4.SPACE.VOLUNIT="" # TYPE: STRING HELP: Volume unit #
 LIM.flcl.CONV.WRITE.FLAM4.SPACE.DATACLASS="" # TYPE: STRING HELP: Data class #
 LIM.flcl.CONV.WRITE.FLAM4.SPACE.MGMTCLASS="" # TYPE: STRING HELP: Management class #
 LIM.flcl.CONV.WRITE.FLAM4.SPACE.STORCLASS="" # TYPE: STRING HELP: Storage class #
 LIM.flcl.CONV.WRITE.FLAM4.SPACE.DISPSTATUS="" # TYPE: NUMBER HELP: Disposition status #
 LIM.flcl.CONV.WRITE.FLAM4.SPACE.NORMALDISP="" # TYPE: NUMBER HELP: Close disposition for  ↩
     normal end #
 LIM.flcl.CONV.WRITE.FLAM4.SPACE.ABORTDISP="" # TYPE: NUMBER HELP: Close disposition for  ↩
     abnormal end #
 LIM.flcl.CONV.WRITE.FLAM4.CMPMODE="" # TYPE: NUMBER HELP: Compression mode [ADC] #
 LIM.flcl.CONV.WRITE.FLAM4.ENCMODE="" # TYPE: NUMBER HELP: Encryption mode [NONE] #
 LIM.flcl.CONV.WRITE.FLAM4.SPLIT="" # TYPE: OVERLAY HELP: Split parameter for FLAM4FILES (z ↩
     /OS only) #
 LIM.flcl.CONV.WRITE.FLAM4.SPLIT.SIZE="" # TYPE: NUMBER HELP: Size for serial splitting ( ↩
     after 1-4096 in megabytes) #
 LIM.flcl.CONV.WRITE.FLAM4.SPLIT.NUMBER="" # TYPE: NUMBER HELP: Number for parallel  ↩
     splitting (2-4 files) #
 LIM.flcl.CONV.WRITE.FLAM4.ORGANIZATION="" # TYPE: NUMBER HELP: Organization of the  ↩
     FLAMFILE [SEQ] #
 LIM.flcl.CONV.WRITE.FLAM4.RECFORMAT="" # TYPE: NUMBER HELP: Record format for the FLAMFILE ↩
      [FIX] #
```

```
LIM.flcl.CONV.WRITE.FLAM4.RECLENGTH="" # TYPE: NUMBER HELP: Record length for the FLAMFILE ↩
     [512] #
LIM.flcl.CONV.WRITE.FLAM4.ORGLENGTH="" # TYPE: NUMBER HELP: Record length to wrap block  ↩
    oriented original data [32752] #
LIM.flcl.CONV.WRITE.FLAM4.KEYPOSITION="" # TYPE: NUMBER HELP: Key position (>=1) for key  ↩
    sequential data sets [1] #
LIM.flcl.CONV.WRITE.FLAM4.KEYLENGTH="" # TYPE: NUMBER HELP: Key length (>=1) for key  ↩
    sequential data sets [8] #
LIM.flcl.CONV.WRITE.FLAM4.COMMENT="" # TYPE: STRING HELP: Comment for the member in the  ↩
    FLAMFILE [internal use] #
LIM.flcl.CONV.WRITE.FLAM4.PASSWORD="" # TYPE: STRING HELP: Passphrase to encrypt the  ↩
    FLAMFILE [''] #
LIM.flcl.CONV.WRITE.FLAM4.KMLIBRARY="" # TYPE: STRING HELP: Library name for FKME [''] #
LIM.flcl.CONV.WRITE.FLAM4.KMFUNCTION="" # TYPE: STRING HELP: Function name of FKME [''] #
LIM.flcl.CONV.WRITE.FLAM4.KMPARAMETER="" # TYPE: STRING HELP: Parameter for the call of  ↩
    FKME [''] #
LIM.flcl.CONV.WRITE.FLAM4.CCSID="" # TYPE: STRING HELP: Conversion to this CCSID [DEFAULT] ↩
     #
LIM.flcl.CONV.WRITE.FLAM4.CASE="" # TYPE: NUMBER HELP: Define upper, lower or special case ↩
     conversion [NONE] #
LIM.flcl.CONV.WRITE.FLAM4.BOM="" # TYPE: SWITCH HELP: Write byte order mark in front of  ↩
    each record [FALSE] #
LIM.flcl.CONV.WRITE.FLAM4.CHRMODE="" # TYPE: NUMBER HELP: Define handling of non- ↩
    convertible characters [STOP] #
LIM.flcl.CONV.WRITE.FLAM4.SUBCHAR="" # TYPE: NUMBER HELP: Default substitution character  ↩
    list [0x1A] (maximal 8 code points) #
LIM.flcl.CONV.WRITE.FLAM4.SYSTABLE="" # TYPE: NUMBER HELP: Define a preloaded system  ↩
    substitution table #
LIM.flcl.CONV.WRITE.FLAM4.USRTABLE="" # TYPE: STRING HELP: Name of the file containing the ↩
     user conversion table #
LIM.flcl.CONV.WRITE.FLAM4.REPORTFILE="" # TYPE: STRING HELP: Name of the report file to  ↩
    log substitutions #
LIM.flcl.CONV.WRITE.FLAM4.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG  ↩
    for write operation #
LIM.flcl.CONV.WRITE.FLAM4.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
    FL_PLATFORM for write operation #
LIM.flcl.CONV.WRITE.FLAM4.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash ↩
     values of output data #
LIM.flcl.CONV.WRITE.FLAM4.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
LIM.flcl.CONV.WRITE.FLAM4.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
LIM.flcl.CONV.WRITE.FLAM4.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string  ↩
    #
LIM.flcl.CONV.WRITE.FLAM4.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
LIM.flcl.CONV.WRITE.FLAM4.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] ↩
     #
LIM.flcl.CONV.WRITE.FLAM4.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG ↩
    ] #
LIM.flcl.CONV.DIR="" # TYPE: OBJECT HELP: Defines how to handle directories and various  ↩
    file types if wildcards used #
LIM.flcl.CONV.DIR.LINK="" # TYPE: SWITCH HELP: Include symbolic links #
LIM.flcl.CONV.DIR.ALIAS="" # TYPE: SWITCH HELP: Resolve data set aliases #
LIM.flcl.CONV.DIR.HIDDEN="" # TYPE: SWITCH HELP: Include hidden files #
LIM.flcl.CONV.DIR.RECURSIVE="" # TYPE: SWITCH HELP: Include sub directories #
LIM.flcl.CONV.DIR.ARCHIVE="" # TYPE: SWITCH HELP: Include archives (FLAMFILEs, ZIP, ...) #
LIM.flcl.CONV.DIR.ONEFILESYSTEM="" # TYPE: SWITCH HELP: Don't cross file system boundaries ↩
     #
LIM.flcl.CONV.DIR.TAPE="" # TYPE: SWITCH HELP: Include files deposited on tapes #
LIM.flcl.CONV.LOGGING="" # TYPE: OVERLAY HELP: Select target for logging #
LIM.flcl.CONV.LOGGING.STREAM="" # TYPE: OBJECT HELP: Log to a stream #
LIM.flcl.CONV.LOGGING.STREAM.FORMAT="" # TYPE: NUMBER HELP: Format of log message [ ↩
```

```
    STANDARD] #
LIM.flcl.CONV.LOGGING.STREAM.IDENT="" # TYPE: STRING HELP: Identifier used to identify log ↩
     entries #
LIM.flcl.CONV.LOGGING.STREAM.STDOUT="" # TYPE: SWITCH HELP: Log messages to STDOUT instead ↩
     of STDERR #
LIM.flcl.CONV.LOGGING.FILE="" # TYPE: OBJECT HELP: Log to a flat file #
LIM.flcl.CONV.LOGGING.FILE.FORMAT="" # TYPE: NUMBER HELP: Format of log message [STANDARD] ↩
     #
LIM.flcl.CONV.LOGGING.FILE.IDENT="" # TYPE: STRING HELP: Identifier used to identify log  ↩
    entries #
LIM.flcl.CONV.LOGGING.FILE.NAME="" # TYPE: STRING HELP: Filename for appending log  ↩
    messages #
LIM.flcl.CONV.LOGGING.SYSTEM="" # TYPE: OBJECT HELP: Use system logging facility #
LIM.flcl.CONV.LOGGING.SYSTEM.IDENT="" # TYPE: STRING HELP: Identifier used to identify log ↩
     entries #
LIM.flcl.CONV.MESSAGE="" # TYPE: OBJECT HELP: Select type of messages to log #
LIM.flcl.CONV.MESSAGE.NONE="" # TYPE: SWITCH HELP: No messages will be logged #
LIM.flcl.CONV.MESSAGE.ERROR="" # TYPE: SWITCH HELP: Error messages will be logged #
LIM.flcl.CONV.MESSAGE.ERRTRACE="" # TYPE: SWITCH HELP: Error trace will be logged #
LIM.flcl.CONV.MESSAGE.WARNING="" # TYPE: SWITCH HELP: Warnings will be logged #
LIM.flcl.CONV.MESSAGE.NOTICE="" # TYPE: SWITCH HELP: Notices will be logged #
LIM.flcl.CONV.MESSAGE.DEBUG="" # TYPE: SWITCH HELP: Debug information will be logged #
LIM.flcl.CONV.MESSAGE.ABOUT="" # TYPE: SWITCH HELP: About information will be logged #
LIM.flcl.CONV.MESSAGE.VERSION="" # TYPE: SWITCH HELP: Version information will be logged #
LIM.flcl.CONV.MESSAGE.INPUT="" # TYPE: SWITCH HELP: Input data will be logged (dangerous  ↩
    for passwords) #
LIM.flcl.CONV.MESSAGE.PARAMETER="" # TYPE: SWITCH HELP: Parsed parameters will be logged #
LIM.flcl.CONV.MESSAGE.STATISTIC="" # TYPE: SWITCH HELP: Statistics will be logged #
LIM.flcl.CONV.MESSAGE.INFO="" # TYPE: SWITCH HELP: Requested information will be logged #
LIM.flcl.CONV.MESSAGE.LICID="" # TYPE: SWITCH HELP: License ID will be logged #
LIM.flcl.CONV.MESSAGE.LICENSE="" # TYPE: SWITCH HELP: License summary will be logged #
LIM.flcl.CONV.MESSAGE.PROGRESS="" # TYPE: SWITCH HELP: Progress bar will be logged #
LIM.flcl.CONV.MESSAGE.ALL="" # TYPE: SWITCH HELP: All messages are logged #
LIM.flcl.CONV.MESSAGE.DEFAULT="" # TYPE: SWITCH HELP: Relevant messages are logged #
LIM.flcl.CONV.MESSAGE.MINIMAL="" # TYPE: SWITCH HELP: Only important messages are logged #
LIM.flcl.CONV.MESSAGE.ERRONLY="" # TYPE: SWITCH HELP: Only errors and error trace are  ↩
    logged #
LIM.flcl.CONV.INVERSE="" # TYPE: STRING HELP: Generate a XCNV command file (xcnv "out= ↩
    thisfile") for inverse conversion [NONE] #
LIM.flcl.XCNV.INPUT="" # TYPE: OBJECT HELP: Inbound parameter #
LIM.flcl.XCNV.INPUT.NET="" # TYPE: OVERLAY HELP: Network [NONE] #
LIM.flcl.XCNV.INPUT.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #
LIM.flcl.XCNV.INPUT.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
LIM.flcl.XCNV.INPUT.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
LIM.flcl.XCNV.INPUT.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
LIM.flcl.XCNV.INPUT.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #
LIM.flcl.XCNV.INPUT.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #
LIM.flcl.XCNV.INPUT.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
LIM.flcl.XCNV.INPUT.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
LIM.flcl.XCNV.INPUT.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
LIM.flcl.XCNV.INPUT.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #
LIM.flcl.XCNV.INPUT.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #
LIM.flcl.XCNV.INPUT.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
LIM.flcl.XCNV.INPUT.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
LIM.flcl.XCNV.INPUT.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
LIM.flcl.XCNV.INPUT.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #
LIM.flcl.XCNV.INPUT.SAV="" # TYPE: OVERLAY HELP: Storage [FILE] #
LIM.flcl.XCNV.INPUT.SAV.FILE="" # TYPE: OBJECT HELP: File handling #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO="" # TYPE: OVERLAY HELP: File IO [BLK] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.BLK="" # TYPE: OBJECT HELP: Read a file block-oriented ( ↩
    mainly for open world) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.BLK.NAME="" # TYPE: STRING HELP: Name/URL of file to read ↩
```

```
     [''==stdin] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.BLK.BLKSIZ="" # TYPE: NUMBER HELP: Block size for read ↩
    operations [AUTO] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.BLK.DATTYP="" # TYPE: NUMBER HELP: Definition of data ↩
    type for better handling [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.BLK.PRNCTR="" # TYPE: NUMBER HELP: Print control ↩
    character handling if read ASA or MCC records [DETACH] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.BLK.FRCBLK="" # TYPE: SWITCH HELP: Enforce block ↩
    orientation on record oriented devices [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.BLK.ADDDLM="" # TYPE: SWITCH HELP: Add a delimiter behind ↩
     each record to produce a text block [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.BLK.SUBSYS="" # TYPE: OBJECT HELP: Specification for FIO ↩
    subsystem (only for z/OS) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.BLK.SUBSYS.NAME="'FLAM'" # TYPE: STRING HELP: Name of the ↩
     subsystem #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.BLK.SUBSYS.PARM="''" # TYPE: STRING HELP: Parameter list ↩
    for the subsystem (comma separated) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.BLK.REMOVE="" # TYPE: SWITCH HELP: Remove original file ↩
    after successful processing [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC="" # TYPE: OBJECT HELP: Read a file record-oriented ( ↩
    mainly for mainframe world) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.NAME="" # TYPE: STRING HELP: Name/URL of file to read ↩
     [''==stdin] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.RECMOD="" # TYPE: NUMBER HELP: Mode used to read ↩
    records [STOP] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.PRNCTR="" # TYPE: NUMBER HELP: Print control ↩
    character handling [DETACH] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.SUPPAD="" # TYPE: SWITCH HELP: Suppress trailing ↩
    padding characters (useful for fixed records) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.PADCHR="" # TYPE: STRING HELP: Define padding ↩
    character for suppression [automatic] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.FILORG="" # TYPE: NUMBER HELP: File organization [SEQ ↩
    ] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.BLKSIZ="" # TYPE: NUMBER HELP: Block size for read ↩
    operations [AUTO] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.RECFMT="" # TYPE: NUMBER HELP: Record format for read ↩
     operations [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.RECLEN="" # TYPE: NUMBER HELP: Record length for read ↩
     operations [512] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.RECCNT="" # TYPE: NUMBER HELP: Amount of records ↩
    handled in one read operation [128] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT="" # TYPE: OVERLAY HELP: Format of the length ↩
    field for open platforms #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.INTEGER="" # TYPE: OBJECT HELP: Use integer ↩
    format for the length field #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.INTEGER.ENDIAN="" # TYPE: NUMBER HELP: Byte ↩
    order [SYSTEM] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.INTEGER.WIDTH="" # TYPE: NUMBER HELP: Bit ↩
    width/rate [B32] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.INTEGER.INCLUDE="" # TYPE: SWITCH HELP: ↩
    Include length field in length value [OFF] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.STRING="" # TYPE: OBJECT HELP: Use string ↩
    format for the length field #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.STRING.CHRSET="" # TYPE: NUMBER HELP: ↩
    Character set [SYSTEM] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.STRING.COUNT="" # TYPE: NUMBER HELP: Character ↩
     count [C06] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.STRING.BASE="" # TYPE: NUMBER HELP: String ↩
    base [B10] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.STRING.INCLUDE="" # TYPE: SWITCH HELP: Include ↩
     length field in length value [OFF] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.BCD="" # TYPE: OBJECT HELP: Use BCD format for ↩
```

```
     the length field #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.BCD.WIDTH="" # TYPE: NUMBER HELP: Bit width/ ↩
    rate [B32] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.BCD.INCLUDE="" # TYPE: SWITCH HELP: Include  ↩
    length field in length value [OFF] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.HOST="" # TYPE: OBJECT HELP: Use host format ( ↩
    LLxx) for the length field #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.HOST.ENDIAN="" # TYPE: NUMBER HELP: Byte order ↩
     [BIG ENDIAN] for first 16 Bit #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.LENFMT.HOST.EXCLUDE="" # TYPE: SWITCH HELP: Record  ↩
    length field not part of the length value [OFF] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.CHRSET="" # TYPE: NUMBER HELP: Character set [auto  ↩
    detection] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.RECDLM="" # TYPE: STRING HELP: Delimiter used to  ↩
    define end of record #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.KEYPOS="" # TYPE: NUMBER HELP: Key position (>=1) for ↩
     index/key sequential data sets [0=NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.KEYLEN="" # TYPE: NUMBER HELP: Key length (>=1) for  ↩
    index/key sequential data sets [if KEYPOS>0 then 8] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.SUBSYS="" # TYPE: OBJECT HELP: Specification for FIO  ↩
    subsystem (only for z/OS) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.SUBSYS.NAME="'FLAM'" # TYPE: STRING HELP: Name of the ↩
     subsystem #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.SUBSYS.PARM="''" # TYPE: STRING HELP: Parameter list  ↩
    for the subsystem (comma separated) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.REC.REMOVE="" # TYPE: SWITCH HELP: Remove original file  ↩
    after successful processing [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.TXT="" # TYPE: OBJECT HELP: Read a file text-oriented ( ↩
    mainly for open world) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.TXT.NAME="" # TYPE: STRING HELP: Name/URL of file to read ↩
     ['']==stdin] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.TXT.RECMOD="" # TYPE: NUMBER HELP: Mode used to read  ↩
    records [STOP] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.TXT.RECLEN="" # TYPE: NUMBER HELP: Record (Host) length / ↩
     Line (Unix/Win) length for read operations [512] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.TXT.RECCNT="" # TYPE: NUMBER HELP: Amount of records  ↩
    handled in one read operation [128] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.TXT.PRNOUT="" # TYPE: NUMBER HELP: Amount of lines per  ↩
    page if print control replacement active [0 = inactive] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.TXT.SUBSYS="" # TYPE: OBJECT HELP: Specification for FIO  ↩
    subsystem (only for z/OS) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.TXT.SUBSYS.NAME="'FLAM'" # TYPE: STRING HELP: Name of the ↩
     subsystem #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.TXT.SUBSYS.PARM="''" # TYPE: STRING HELP: Parameter list  ↩
    for the subsystem (comma separated) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.TXT.REMOVE="" # TYPE: SWITCH HELP: Remove original file  ↩
    after successful processing [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4="" # TYPE: OBJECT HELP: Read a FLAMFILE record- ↩
    oriented #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.NAME="" # TYPE: STRING HELP: Name/URL of FLAMFILE to  ↩
    read [required] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.MEMBER="" # TYPE: STRING HELP: Name or index of the  ↩
    member in the FLAMFILE #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.RECLEN="" # TYPE: NUMBER HELP: Maximal record length  ↩
    for read operations [512] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.RECCNT="" # TYPE: NUMBER HELP: Amount of records  ↩
    handled in one read operation [128] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.TRUCAT="" # TYPE: SWITCH HELP: Allow truncation of  ↩
    records [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.NONFIL="" # TYPE: SWITCH HELP: Switch to replace  ↩
    original file name with FILEnnnn [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.PRNCTR="" # TYPE: NUMBER HELP: Print control  ↩
```

```
    character handling [DETACH] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.DEVICE="" # TYPE: NUMBER HELP: Device used to store ↩
    the FLAMFILE [FLATFILE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.PASSWD="" # TYPE: STRING HELP: Passphrase to decrypt ↩
    the FLAMFILE [''] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.KMELIB="" # TYPE: STRING HELP: Library name for FKME ↩
    [''] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.KMEFUC="" # TYPE: STRING HELP: Function name of FKME ↩
    [''] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.KMEPAR="" # TYPE: STRING HELP: Parameter for the call ↩
     of FKME [''] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.EXTCMP="" # TYPE: STRING HELP: Name of user exit for ↩
    compression #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.EXTDCO="" # TYPE: STRING HELP: Name of user exit for ↩
    decompression #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.SECINF="" # TYPE: NUMBER HELP: Define handling of ↩
    secure information #
LIM.flcl.XCNV.INPUT.SAV.FILE.FIO.FL4.REMOVE="" # TYPE: SWITCH HELP: Remove FLAMFILE after ↩
    successful processing [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV="" # TYPE: OVERLAY HELP: Conversion [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BLK="" # TYPE: OBJECT HELP: Convert a record list to one ↩
    block (binary or text) of data #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BLK.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size ↩
    for preallocation [65536] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BLK.METHOD="" # TYPE: NUMBER HELP: Method for block/text ↩
    formatting [SYSTEM] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BLK.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM] ↩
     #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BLK.SUPTWS="" # TYPE: SWITCH HELP: Suppress trailing ↩
    whitespace [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BLK.RPLTAB="" # TYPE: NUMBER HELP: Replace horizontal ↩
    tabulators by spaces using this tab width [0 = no replacement] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BLK.RPLVTB="" # TYPE: NUMBER HELP: Replace vertical ↩
    tabulators by new lines using this tab width [0 = no replacement] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BLK.RPLCTR="" # TYPE: NUMBER HELP: Replace remaining ↩
    control characters [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC="" # TYPE: OBJECT HELP: Convert a block (binary or ↩
    text) of data into a list of records #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC.METHOD="" # TYPE: NUMBER HELP: Method for record/text ↩
     formatting [DEFAULT] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM] ↩
     #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds, ↩
    filling rest of page with blank lines assuming n lines per page [0 = no replacement] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC.RPLTAB="" # TYPE: NUMBER HELP: Replace horizontal ↩
    tabulators by spaces using this tab width [0 = no replacement] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC.RPLVTB="" # TYPE: NUMBER HELP: Replace vertical ↩
    tabulators by new lines using this tab width [0 = no replacement] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC.RPLCTR="" # TYPE: NUMBER HELP: Replace remaining ↩
    control characters [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC.SUPTWS="" # TYPE: SWITCH HELP: Suppress trailing ↩
    whitespaces [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC.NELDLM="" # TYPE: SWITCH HELP: Activate NEL (0x85) as ↩
     delimiter for ASCII character sets [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC.PADCHR="" # TYPE: NUMBER HELP: Padding character [0 ↩
    x00] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC.RECLEN="" # TYPE: NUMBER HELP: Length used to cut the ↩
     data block in records [512] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size ↩
    for preallocation [65536] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC.INICNT="" # TYPE: NUMBER HELP: Initial amount of ↩
    records for preallocation [128] #
```

```
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.REC.SKPTXT="" # TYPE: SWITCH HELP: Convert only if record ↩
     length fields found in the data [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.GZP="" # TYPE: OBJECT HELP: Read ZLIB/GZIP/RCF1950-2  ↩
    compressed data #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.GZP.MEMBER="" # TYPE: STRING HELP: Index of the member in ↩
     the GZIP-File to read #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.GZP.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation[65536] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BZ2="" # TYPE: OBJECT HELP: Read BZIP2 compressed data #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BZ2.MEMBER="" # TYPE: STRING HELP: Index of the member in ↩
     the BZIP2-File to read #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BZ2.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BZ2.SMLSPC="" # TYPE: SWITCH HELP: Use small memory space ↩
     for decompression [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.LXZ="" # TYPE: OBJECT HELP: Read XZ (LZMA) compressed  ↩
    data #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.LXZ.MEMBER="" # TYPE: STRING HELP: Index of the member in ↩
     the XT-File to read #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.LXZ.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.ZIP="" # TYPE: OBJECT HELP: Read compressed data (GZIP,  ↩
    BZIP2, LZMA, ...) #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.ZIP.MEMBER="" # TYPE: STRING HELP: Index of the member in ↩
     the compressed file to read #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.ZIP.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation[65536] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR="" # TYPE: OBJECT HELP: Convert character sets (ASCII ↩
    , EBCDIC, UNICODE) #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.RECCNT="" # TYPE: NUMBER HELP: Initial amount of  ↩
    records for preallocation [128] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.METHOD="" # TYPE: NUMBER HELP: Enforces record or  ↩
    block-oriented conversion [AUTO] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.CASMOD="" # TYPE: NUMBER HELP: Define upper, lower or ↩
     special case conversion [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.FROM="" # TYPE: STRING HELP: Conversion from this  ↩
    CCSID [auto detection] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.MODE="" # TYPE: NUMBER HELP: Define handling of non- ↩
    convertible characters #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.TO="" # TYPE: STRING HELP: Conversion to this CCSID [ ↩
    UTF-8] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.WRTBOM="" # TYPE: SWITCH HELP: Write byte order mark  ↩
    in front of the output text #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.HDLBOM="" # TYPE: SWITCH HELP: Handle byte order  ↩
    change if get wrong BOM in the middle of data #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new  ↩
    line (0x15) by line feed (0x25) on input #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.ELF2NL="" # TYPE: SWITCH HELP: Replace EBCDIC line  ↩
    feed (0x25) by new line (0x15) on output #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.SUBCHR="" # TYPE: NUMBER HELP: Default substitution  ↩
    character list [0x1A] (maximal 8 code points) #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.SYSTAB="" # TYPE: NUMBER HELP: Define a preloaded  ↩
    system substitution table #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.USRTAB="" # TYPE: STRING HELP: Name of the file  ↩
    containing the user conversion table #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.REPFIL="" # TYPE: STRING HELP: Name of the report  ↩
    file to log substitutions #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.SKPBIN="" # TYPE: SWITCH HELP: Skip character  ↩
    conversion if binary data is detected [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.CHR.SKPEQU="" # TYPE: SWITCH HELP: Skip character  ↩
```

```
    conversion if from-code equals to-code [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BAS="" # TYPE: OBJECT HELP: Convert Base16/32/64/85 ↩
    encoded data to binary data #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BAS.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size ↩
    for preallocation [65536] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BAS.RECCNT="" # TYPE: NUMBER HELP: Initial amount of ↩
    records for preallocation [128] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BAS.METHOD="" # TYPE: NUMBER HELP: Base encoding selector ↩
     [AUTO] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BAS.MODE="" # TYPE: NUMBER HELP: Enforces record or block ↩
    -oriented conversion [BLOCK] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BAS.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM] ↩
     #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BAS.CRCCHK="" # TYPE: SWITCH HELP: Check CRC if available ↩
     [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BAS.ARMCHK="" # TYPE: SWITCH HELP: Check ARMOR trailer if ↩
     available [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BAS.RSTCHK="" # TYPE: SWITCH HELP: Check if a rest after ↩
    encoded data [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.BAS.SKPNOB="" # TYPE: SWITCH HELP: Skip base decoding if ↩
    no base encoding detected [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.HSH="" # TYPE: OBJECT HELP: Generation or verification of ↩
     one way hash values #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.HSH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.HSH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum ↩
     verification #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.HSH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex- ↩
    string #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.HSH.CHECK.FILE="" # TYPE: STRING HELP: File which ↩
    contains Checksum as hex-string #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.HSH.FORMAT="" # TYPE: NUMBER HELP: Define printout format ↩
     [GNU] #
LIM.flcl.XCNV.INPUT.SAV.FILE.CNV.HSH.OUTPUT="" # TYPE: STRING HELP: Name of the output ↩
    file [LOG] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT="" # TYPE: OVERLAY HELP: Formatting [BIN] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BIN="" # TYPE: OBJECT HELP: Format data in binary ↩
    elements (block/records) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BIN.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size ↩
    for preallocation [65536] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BIN.RECCNT="" # TYPE: NUMBER HELP: Initial amount of ↩
    records for preallocation [128] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BIN.TERASE="" # TYPE: SWITCH HELP: Erase data from memory ↩
     when no longer needed [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BIN.SEGINT="" # TYPE: NUMBER HELP: Segment integrity mode ↩
     [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BIN.LENMOD="" # TYPE: NUMBER HELP: Compression mode for ↩
    length fields [COPY] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BIN.LENCNF="" # TYPE: NUMBER HELP: Encryption mode for ↩
    length fields [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BIN.DATMOD="" # TYPE: NUMBER HELP: Compression mode for ↩
    data [COPY] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BIN.DATCNF="" # TYPE: NUMBER HELP: Encryption mode for ↩
    data [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BLK="" # TYPE: OBJECT HELP: Format data in one block ↩
    element (all meta data lost) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BLK.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size ↩
    for preallocation [65536] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BLK.TERASE="" # TYPE: SWITCH HELP: Erase data from memory ↩
     when no longer needed [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BLK.SEGINT="" # TYPE: NUMBER HELP: Segment integrity mode ↩
     [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BLK.LENMOD="" # TYPE: NUMBER HELP: Compression mode for ↩
```

```
    length fields [COPY] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BLK.LENCNF="" # TYPE: NUMBER HELP: Encryption mode for ↩
    length fields [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BLK.DATMOD="" # TYPE: NUMBER HELP: Compression mode for ↩
    data [COPY] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.BLK.DATCNF="" # TYPE: NUMBER HELP: Encryption mode for ↩
    data [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.REC="" # TYPE: OBJECT HELP: Format data in several ↩
    wrapped record elements (all meta data lost) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.REC.METHOD="" # TYPE: NUMBER HELP: Method for record ↩
    formatting [AUTO] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.REC.WRPERR="" # TYPE: SWITCH HELP: If no method defined ( ↩
    auto detection) enforce an error if no length format (wrapping) detected [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.REC.RECLEN="" # TYPE: NUMBER HELP: Maximum length of a ↩
    record [512] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.REC.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size ↩
    for preallocation [65536] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.REC.INICNT="" # TYPE: NUMBER HELP: Initial amount of ↩
    records for preallocation [128] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.REC.TERASE="" # TYPE: SWITCH HELP: Erase data from memory ↩
     when no longer needed [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.REC.SEGINT="" # TYPE: NUMBER HELP: Segment integrity mode ↩
     [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.REC.LENMOD="" # TYPE: NUMBER HELP: Compression mode for ↩
    length fields [COPY] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.REC.LENCNF="" # TYPE: NUMBER HELP: Encryption mode for ↩
    length fields [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.REC.DATMOD="" # TYPE: NUMBER HELP: Compression mode for ↩
    data [COPY] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.REC.DATCNF="" # TYPE: NUMBER HELP: Encryption mode for ↩
    data [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT="" # TYPE: OBJECT HELP: Format data based on ↩
    delimiters in text elements (record and rest) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.METHOD="" # TYPE: NUMBER HELP: Method for text ↩
    formatting [DEFAULT] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.CHRSET="" # TYPE: NUMBER HELP: Character set [auto ↩
    detection] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds, ↩
    filling rest of page with blank lines assuming n lines per page [0 = no replacement] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.SUPTWS="" # TYPE: SWITCH HELP: Suppress trailing ↩
    whitespaces [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.NELDLM="" # TYPE: SWITCH HELP: Activate NEL (0x85) as ↩
     delimiter for ASCII character sets [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.PADCHR="" # TYPE: NUMBER HELP: Padding character (is ↩
    regarded as additional delimiter) [0x00] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.RECLEN="" # TYPE: NUMBER HELP: Maximum length of a ↩
    text record (Host) / line (Unix/Win) [512] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size ↩
    for preallocation [65536] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.INICNT="" # TYPE: NUMBER HELP: Initial amount of ↩
    records for preallocation [128] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.BINERR="" # TYPE: SWITCH HELP: Enforce an error if ↩
    binary data detected [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.CHRERR="" # TYPE: SWITCH HELP: Enforce an error if ↩
    character data without delimiter detected [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.TERASE="" # TYPE: SWITCH HELP: Erase data from memory ↩
     when no longer needed [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.SEGINT="" # TYPE: NUMBER HELP: Segment integrity mode ↩
     [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.LENMOD="" # TYPE: NUMBER HELP: Compression mode for ↩
    length fields [COPY] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.LENCNF="" # TYPE: NUMBER HELP: Encryption mode for ↩
```

```
    length fields [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.DATMOD="" # TYPE: NUMBER HELP: Compression mode for  ↩
    data [COPY] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.TXT.DATCNF="" # TYPE: NUMBER HELP: Encryption mode for  ↩
    data [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.XML="" # TYPE: OBJECT HELP: Format data stream in XML  ↩
    elements (tags, attributes, data, ...) #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.XML.CHRSET="" # TYPE: NUMBER HELP: Character set [auto] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.XML.FMTERR="" # TYPE: SWITCH HELP: Enforce a format error ↩
     if data contains only space or is empty [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.XML.NOCMNT="" # TYPE: SWITCH HELP: Ignore XML comments [ ↩
    FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.XML.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.XML.INICNT="" # TYPE: NUMBER HELP: Initial amount of  ↩
    records for preallocation [128] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.XML.TERASE="" # TYPE: SWITCH HELP: Erase data from memory ↩
     when no longer needed [FALSE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.XML.SEGINT="" # TYPE: NUMBER HELP: Segment integrity mode ↩
     [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.XML.LENMOD="" # TYPE: NUMBER HELP: Compression mode for  ↩
    length fields [COPY] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.XML.LENCNF="" # TYPE: NUMBER HELP: Encryption mode for  ↩
    length fields [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.XML.DATMOD="" # TYPE: NUMBER HELP: Compression mode for  ↩
    data [COPY] #
LIM.flcl.XCNV.INPUT.SAV.FILE.FMT.XML.DATCNF="" # TYPE: NUMBER HELP: Encryption mode for  ↩
    data [NONE] #
LIM.flcl.XCNV.INPUT.SAV.FILE.ENV="" # TYPE: OBJECT HELP: Environment #
LIM.flcl.XCNV.INPUT.SAV.FILE.ENV.KEYWORD="" # TYPE: STRING HELP: Keyword of environment  ↩
    variable #
LIM.flcl.XCNV.INPUT.SAV.FILE.ENV.VALUE="" # TYPE: STRING HELP: Value of environment  ↩
    variable #
LIM.flcl.XCNV.OUTPUT="" # TYPE: OBJECT HELP: Outbound parameter #
LIM.flcl.XCNV.OUTPUT.NET="" # TYPE: OVERLAY HELP: Network [NONE] #
LIM.flcl.XCNV.OUTPUT.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #
LIM.flcl.XCNV.OUTPUT.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
LIM.flcl.XCNV.OUTPUT.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
LIM.flcl.XCNV.OUTPUT.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
LIM.flcl.XCNV.OUTPUT.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #
LIM.flcl.XCNV.OUTPUT.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #
LIM.flcl.XCNV.OUTPUT.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
LIM.flcl.XCNV.OUTPUT.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
LIM.flcl.XCNV.OUTPUT.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
LIM.flcl.XCNV.OUTPUT.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #
LIM.flcl.XCNV.OUTPUT.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #
LIM.flcl.XCNV.OUTPUT.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
LIM.flcl.XCNV.OUTPUT.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
LIM.flcl.XCNV.OUTPUT.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
LIM.flcl.XCNV.OUTPUT.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #
LIM.flcl.XCNV.OUTPUT.SAV="" # TYPE: OVERLAY HELP: Storage [FILE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE="" # TYPE: OBJECT HELP: File handling #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT="" # TYPE: OVERLAY HELP: Formatting [BIN] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.BIN="" # TYPE: OBJECT HELP: Format elements as binary  ↩
    data (block/records) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.BIN.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.BIN.RECCNT="" # TYPE: NUMBER HELP: Initial amount of  ↩
    records for preallocation [128] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.BLK="" # TYPE: OBJECT HELP: Format all elements to one  ↩
    block of data (all meta data lost) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.BLK.METHOD="" # TYPE: NUMBER HELP: Method for block  ↩
```

```
    formatting [JOIN ELEMENTS TOGETHER] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.BLK.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.REC="" # TYPE: OBJECT HELP: Format each element to one  ↩
    standard record (possible meta data lost) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.REC.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.REC.INICNT="" # TYPE: NUMBER HELP: Initial amount of  ↩
    records for preallocation [128] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.TXT="" # TYPE: OBJECT HELP: Format elements as text data ↩
     with delimiters #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.TXT.METHOD="" # TYPE: NUMBER HELP: Method for text  ↩
    formatting [DEFAULT] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.TXT.CHRSET="" # TYPE: NUMBER HELP: Character set [auto]  ↩
    #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.TXT.RPLTAB="" # TYPE: NUMBER HELP: Replace horizontal  ↩
    tabulators by spaces using this tab width [0 - no replacement] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.TXT.RPLVTB="" # TYPE: NUMBER HELP: Replace vertical  ↩
    tabulators by new lines using this tab width [0 = no replacement] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.TXT.RPLCTR="" # TYPE: NUMBER HELP: Replace remaining  ↩
    control characters [NONE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.TXT.SUPTWS="" # TYPE: SWITCH HELP: Suppress trailing  ↩
    whitespaces [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.TXT.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.TXT.INICNT="" # TYPE: NUMBER HELP: Initial amount of  ↩
    records for preallocation [128] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML="" # TYPE: OBJECT HELP: Format elements (tags,  ↩
    attributes, data, ...) as XML data stream #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD="" # TYPE: OVERLAY HELP: Method for XML  ↩
    formatting [STANDARD] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.STANDARD="" # TYPE: OBJECT HELP: Reproduce  ↩
    all XML elements as is #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.STANDARD.NOCMNT="" # TYPE: SWITCH HELP: Do  ↩
    not write out XML comments [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.STANDARD.BUFSIZ="" # TYPE: NUMBER HELP:  ↩
    Initial buffer size for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.STANDARD.INICNT="" # TYPE: NUMBER HELP:  ↩
    Initial amount of records for preallocation [128] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.PRETTYPRINT="" # TYPE: OBJECT HELP:  ↩
    Indentation of XML tags for human readability #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.PRETTYPRINT.MAXWSP="" # TYPE: NUMBER HELP:  ↩
    Size of internal whitespace cache (only used when method is minimize or pretty printing ↩
    ) [4096] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.PRETTYPRINT.WSPERR="" # TYPE: NUMBER HELP:  ↩
    Action when the whitespace cache overflows [WRITEOUT] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.PRETTYPRINT.INDSIZ="-1" # TYPE: NUMBER HELP:  ↩
    Number of indentation characters per indentation level [2 (SPACE) or 1 (TAB)] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.PRETTYPRINT.INDCHR="" # TYPE: NUMBER HELP:  ↩
    Indentation character used to indent XML tags [SPACE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.PRETTYPRINT.NOCMNT="" # TYPE: SWITCH HELP: Do ↩
     not write out XML comments [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.PRETTYPRINT.BUFSIZ="" # TYPE: NUMBER HELP:  ↩
    Initial buffer size for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.PRETTYPRINT.INICNT="" # TYPE: NUMBER HELP:  ↩
    Initial amount of elements for preallocation [128] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.MINIMIZED="" # TYPE: OBJECT HELP: Discards  ↩
    comments, insignificant whitespace and whitespace-only sections between tags #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.MINIMIZED.MAXWSP="" # TYPE: NUMBER HELP: Size ↩
     of internal whitespace cache (only used when method is minimize or pretty printing)  ↩
    [4096] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.MINIMIZED.WSPERR="" # TYPE: NUMBER HELP:  ↩
```

```
    Action when the whitespace cache overflows [WRITEOUT] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.MINIMIZED.BUFSIZ="" # TYPE: NUMBER HELP:  ↩
    Initial buffer size for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.MINIMIZED.INICNT="" # TYPE: NUMBER HELP:  ↩
    Initial amount of elements for preallocation [128] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.DUMP="" # TYPE: OBJECT HELP: Dumps all  ↩
    elements in raw XML format (also useful for non XML elements) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.DUMP.DATAFMT="" # TYPE: NUMBER HELP: Format  ↩
    in which element data is dumped [HEX] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.DUMP.ATTRFMT="" # TYPE: NUMBER HELP: Format  ↩
    in which attribute data is dumped [HEX] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.DUMP.HASHFMT="" # TYPE: NUMBER HELP: Format  ↩
    in which hash data is dumped [HEX] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.DUMP.INDSIZ="-1" # TYPE: NUMBER HELP: Number  ↩
    of indentation characters per indentation level [2 (SPACE) or 1 (TAB)] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.DUMP.INDCHR="" # TYPE: NUMBER HELP:  ↩
    Indentation character used to indent XML tags [SPACE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.DUMP.BUFSIZ="" # TYPE: NUMBER HELP: Initial  ↩
    buffer size for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FMT.XML.METHOD.DUMP.INICNT="" # TYPE: NUMBER HELP: Initial  ↩
    amount of elements for preallocation [128] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV="" # TYPE: OVERLAY HELP: Conversion [NONE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BLK="" # TYPE: OBJECT HELP: Convert a record list to one ↩
     block (binary or text) of data #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BLK.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BLK.METHOD="" # TYPE: NUMBER HELP: Method for block/text ↩
     formatting [SYSTEM] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BLK.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM ↩
    ] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BLK.SUPTWS="" # TYPE: SWITCH HELP: Suppress trailing  ↩
    whitespace [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BLK.RPLTAB="" # TYPE: NUMBER HELP: Replace horizontal  ↩
    tabulators by spaces using this tab width [0 = no replacement] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BLK.RPLVTB="" # TYPE: NUMBER HELP: Replace vertical  ↩
    tabulators by new lines using this tab width [0 = no replacement] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BLK.RPLCTR="" # TYPE: NUMBER HELP: Replace remaining  ↩
    control characters [NONE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.REC="" # TYPE: OBJECT HELP: Convert a block (binary or  ↩
    text) of data into a list of records #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.REC.METHOD="" # TYPE: NUMBER HELP: Method for record/ ↩
    text formatting [DEFAULT] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.REC.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM ↩
    ] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.REC.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds,  ↩
    filling rest of page with blank lines assuming n lines per page [0 = no replacement] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.REC.RPLTAB="" # TYPE: NUMBER HELP: Replace horizontal  ↩
    tabulators by spaces using this tab width [0 = no replacement] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.REC.RPLVTB="" # TYPE: NUMBER HELP: Replace vertical  ↩
    tabulators by new lines using this tab width [0 = no replacement] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.REC.RPLCTR="" # TYPE: NUMBER HELP: Replace remaining  ↩
    control characters [NONE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.REC.SUPTWS="" # TYPE: SWITCH HELP: Suppress trailing  ↩
    whitespaces [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.REC.NELDLM="" # TYPE: SWITCH HELP: Activate NEL (0x85)  ↩
    as delimiter for ASCII character sets [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.REC.PADCHR="" # TYPE: NUMBER HELP: Padding character [0 ↩
    x00] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.REC.RECLEN="" # TYPE: NUMBER HELP: Length used to cut  ↩
    the data block in records [512] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.REC.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
```

```
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.REC.INICNT="" # TYPE: NUMBER HELP: Initial amount of records for preallocation [128] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.GZP="" # TYPE: OBJECT HELP: Write GZIP/RCF1952 compressed data #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.GZP.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.GZP.CMPLEV="" # TYPE: NUMBER HELP: Compression level [ AUTO] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.GZP.TXTDAT="" # TYPE: SWITCH HELP: Mark content as text data in header [from origin] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.GZP.MODTIM="" # TYPE: NUMBER HELP: Modification time in header (seconds since 1970) [from origin] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.GZP.OPRSYS="" # TYPE: NUMBER HELP: Operation/file system in header [from origin] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.GZP.EXTRAD="" # TYPE: STRING HELP: Extra data for header #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.GZP.MBRNAM="" # TYPE: STRING HELP: Rename filename for header [from origin] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.GZP.COMENT="" # TYPE: STRING HELP: Comment for header #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.GZP.HDRCHK="" # TYPE: SWITCH HELP: Use checksum for header [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BZ2="" # TYPE: OBJECT HELP: Write BZIP2 compressed data #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BZ2.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BZ2.CMPLEV="" # TYPE: NUMBER HELP: Compression level, defines block size for compression strength [AUTO] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BZ2.WRKFAC="" # TYPE: NUMBER HELP: Work factor from 0 to 250 for highly repetitive data handling [0==30] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.LXZ="" # TYPE: OBJECT HELP: Write XZ (LZMA) compressed data #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.LXZ.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.LXZ.CMPLEV="" # TYPE: NUMBER HELP: Compression level [ AUTO] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR="" # TYPE: OBJECT HELP: Convert character sets ( ASCII, EBCDIC, UNICODE) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.RECCNT="" # TYPE: NUMBER HELP: Initial amount of records for preallocation [128] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.METHOD="" # TYPE: NUMBER HELP: Enforces record or block-oriented conversion [AUTO] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.CASMOD="" # TYPE: NUMBER HELP: Define upper, lower or special case conversion [NONE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.FROM="" # TYPE: STRING HELP: Conversion from this CCSID [DEFAULT] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.MODE="" # TYPE: NUMBER HELP: Define handling of non- convertible characters #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.TO="" # TYPE: STRING HELP: Conversion to this CCSID [DEFAULT] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.WRTBOM="" # TYPE: SWITCH HELP: Write byte order mark in front of the output text #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.HDLBOM="" # TYPE: SWITCH HELP: Handle byte order change if get wrong BOM in the middle of data #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15) by line feed (0x25) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.ELF2NL="" # TYPE: SWITCH HELP: Replace EBCDIC line feed (0x25) by new line (0x15) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.SUBCHR="" # TYPE: NUMBER HELP: Default substitution character list [0x1A] (maximal 8 code points) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.SYSTAB="" # TYPE: NUMBER HELP: Define a preloaded
```

```
    system substitution table #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.USRTAB="" # TYPE: STRING HELP: Name of the file  ↩
    containing the user conversion table #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.REPFIL="" # TYPE: STRING HELP: Name of the report  ↩
    file to log substitutions #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.SKPBIN="" # TYPE: SWITCH HELP: Skip character  ↩
    conversion if binary data is detected [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.CHR.SKPEQU="" # TYPE: SWITCH HELP: Skip character  ↩
    conversion if from-code equals to-code [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BAS="" # TYPE: OBJECT HELP: Convert binary data to  ↩
    Base16/32/64/85 encoded data #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BAS.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BAS.RECCNT="" # TYPE: NUMBER HELP: Initial amount of  ↩
    records for preallocation [128] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BAS.METHOD="" # TYPE: NUMBER HELP: Base encoding  ↩
    selector [BASE64] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BAS.MODE="" # TYPE: NUMBER HELP: Enforces record or  ↩
    block-oriented conversion [BLOCK] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BAS.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM ↩
    ] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BAS.LINE="" # TYPE: NUMBER HELP: Maximum length of an  ↩
    encoded line when mode is BLOCK [0=unlimited] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BAS.DELIM="" # TYPE: NUMBER HELP: Line delimiter to use  ↩
    when mode is BLOCK and linlen>0 [SYSTEM] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BAS.CRCCHK="" # TYPE: SWITCH HELP: Add CRC checksum to  ↩
    the base encoded data [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BAS.ARMOR="" # TYPE: OBJECT HELP: Activate ARMOR  ↩
    encoding and defines corresponding parameter #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BAS.ARMOR.HEADER="" # TYPE: STRING HELP: String for the  ↩
    header [AUTO] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BAS.ARMOR.VERSION="" # TYPE: STRING HELP: Version string ↩
     as key value [AUTO] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.BAS.ARMOR.COMMENT="" # TYPE: STRING HELP: Comment string ↩
     as key value [AUTO] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.HSH="" # TYPE: OBJECT HELP: Generation or verification  ↩
    of one way hash values #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.HSH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.HSH.CHECK="" # TYPE: OVERLAY HELP: Parameter for  ↩
    checksum verification #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.HSH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex- ↩
    string #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.HSH.CHECK.FILE="" # TYPE: STRING HELP: File which  ↩
    contains Checksum as hex-string #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.HSH.FORMAT="" # TYPE: NUMBER HELP: Define printout  ↩
    format [GNU] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.CNV.HSH.OUTPUT="" # TYPE: STRING HELP: Name of the output  ↩
    file [LOG] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO="" # TYPE: OVERLAY HELP: File IO [BLK] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK="" # TYPE: OBJECT HELP: Write a file block-oriented  ↩
    (mainly open world) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.NAME="" # TYPE: STRING HELP: Name/URL of file to  ↩
    write [''==stdout] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC="" # TYPE: OBJECT HELP: Platform dependent  ↩
    file allocation parameter #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.ORGANIZATION="" # TYPE: NUMBER HELP: File  ↩
    organization [SEQ] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.BLKSIZE="" # TYPE: NUMBER HELP: Block size [ ↩
    system] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.RECFORMAT="" # TYPE: NUMBER HELP: Record  ↩
    format [auto] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.RECLENGTH="" # TYPE: NUMBER HELP: Record  ↩
```

```
    length (data only, no print control and length fields) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.KEYDISP="" # TYPE: NUMBER HELP: Key  ↩
    disposition for index/key sequential data sets [OLD] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.KEYPOSITION="" # TYPE: NUMBER HELP: Key  ↩
    position (>=1) for index/key sequential data sets [1] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.KEYLENGTH="" # TYPE: NUMBER HELP: Key length  ↩
    (>=1) for index/key sequential data sets [8] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SUBSYSTEM="" # TYPE: OBJECT HELP:  ↩
    Specification for FIO subsystem (only for z/OS) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP:  ↩
    Name of the subsystem #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SUBSYSTEM.PARM="'" # TYPE: STRING HELP:  ↩
    Parameter list for the subsystem (comma separated) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE="" # TYPE: OBJECT HELP: Platform  ↩
    dependent space definition parameter #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.PRIMARY="" # TYPE: NUMBER HELP: Primary ↩
     size in megabytes #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.SECONDARY="" # TYPE: NUMBER HELP:  ↩
    Secondary size in megabytes #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.DIRBLOCKS="" # TYPE: NUMBER HELP:  ↩
    Directory blocks for partitioned data sets #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.VOLSER="" # TYPE: STRING HELP: Volume  ↩
    serial number #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.VOLUNIT="" # TYPE: STRING HELP: Volume  ↩
    unit #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.DATACLASS="" # TYPE: STRING HELP: Data  ↩
    class #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.MGMTCLASS="" # TYPE: STRING HELP:  ↩
    Management class #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.STORCLASS="" # TYPE: STRING HELP:  ↩
    Storage class #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.NORMALDISP="" # TYPE: NUMBER HELP:  ↩
    Close disposition for normal end [CATALOG] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.ABORTDISP="" # TYPE: NUMBER HELP: Close ↩
     disposition for abnormal end [DELETE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.RELEASE="" # TYPE: NUMBER HELP: Release ↩
     unused space when file is closed [SEQ=YES, PDS=NO] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.UNMOVABLE="" # TYPE: SWITCH HELP:  ↩
    Allocate space unmovable [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.CONTIG="" # TYPE: SWITCH HELP: Allocate ↩
     space contiguously [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.PERMANENT="" # TYPE: SWITCH HELP:  ↩
    Allocate space permanently [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.SPACE.RLS="" # TYPE: NUMBER HELP: Specifies  ↩
    the type of record level sharing (RLS) being done for a specific data set [NONE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.LIKE="" # TYPE: STRING HELP: Specifies the  ↩
    name of the model data set from which the attributes are to be copied #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FALLOC.RENEW="" # TYPE: SWITCH HELP: Enforce new  ↩
    allocation by a remove of an existing file #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.REMAIN="" # TYPE: SWITCH HELP: Remain existing files ↩
     (don't overwrite) [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.APPEND="" # TYPE: SWITCH HELP: Append data to this  ↩
    file [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FLUSH="" # TYPE: SWITCH HELP: Enforce flush in front ↩
     of close file [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.FRCREC="" # TYPE: SWITCH HELP: Enforce record  ↩
    orientation on record oriented devices [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.ADDDLM="" # TYPE: SWITCH HELP: Add a delimiter  ↩
    behind each record to produce a text block [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.ACSTIM="" # TYPE: NUMBER HELP: Define the last  ↩
    access time for this file #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.MODTIM="" # TYPE: NUMBER HELP: Define the last  ↩
```

```
    modification time for this file #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.PRNCTR="" # TYPE: NUMBER HELP: Print control  ↩
    character handling if write ASA or MCC records [DETACH] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.BLK.BINDMP="" # TYPE: SWITCH HELP: Write dump for binary ↩
     data [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC="" # TYPE: OBJECT HELP: Write a file record-oriented ↩
     (mainly mainframe world) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.NAME="" # TYPE: STRING HELP: Name/URL of file to  ↩
    write [''==stdout] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC="" # TYPE: OBJECT HELP: Platform dependent  ↩
    file allocation parameter #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.ORGANIZATION="" # TYPE: NUMBER HELP: File  ↩
    organization [SEQ] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.BLKSIZE="" # TYPE: NUMBER HELP: Block size [ ↩
    system] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.RECFORMAT="" # TYPE: NUMBER HELP: Record  ↩
    format [auto] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.RECLENGTH="" # TYPE: NUMBER HELP: Record  ↩
    length (data only, no print control and length fields) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.KEYDISP="" # TYPE: NUMBER HELP: Key  ↩
    disposition for index/key sequential data sets [OLD] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.KEYPOSITION="" # TYPE: NUMBER HELP: Key  ↩
    position (>=1) for index/key sequential data sets [1] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.KEYLENGTH="" # TYPE: NUMBER HELP: Key length  ↩
    (>=1) for index/key sequential data sets [8] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SUBSYSTEM="" # TYPE: OBJECT HELP:  ↩
    Specification for FIO subsystem (only for z/OS) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP:  ↩
    Name of the subsystem #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SUBSYSTEM.PARM="'" # TYPE: STRING HELP:  ↩
    Parameter list for the subsystem (comma separated) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE="" # TYPE: OBJECT HELP: Platform  ↩
    dependent space definition parameter #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.PRIMARY="" # TYPE: NUMBER HELP: Primary ↩
     size in megabytes #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.SECONDARY="" # TYPE: NUMBER HELP:  ↩
    Secondary size in megabytes #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.DIRBLOCKS="" # TYPE: NUMBER HELP:  ↩
    Directory blocks for partitioned data sets #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.VOLSER="" # TYPE: STRING HELP: Volume  ↩
    serial number #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.VOLUNIT="" # TYPE: STRING HELP: Volume  ↩
    unit #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.DATACLASS="" # TYPE: STRING HELP: Data  ↩
    class #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.MGMTCLASS="" # TYPE: STRING HELP:  ↩
    Management class #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.STORCLASS="" # TYPE: STRING HELP:  ↩
    Storage class #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.NORMALDISP="" # TYPE: NUMBER HELP:  ↩
    Close disposition for normal end [CATALOG] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.ABORTDISP="" # TYPE: NUMBER HELP: Close ↩
     disposition for abnormal end [DELETE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.RELEASE="" # TYPE: NUMBER HELP: Release ↩
     unused space when file is closed [SEQ=YES, PDS=NO] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.UNMOVABLE="" # TYPE: SWITCH HELP:  ↩
    Allocate space unmovable [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.CONTIG="" # TYPE: SWITCH HELP: Allocate ↩
     space contiguously [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.PERMANENT="" # TYPE: SWITCH HELP:  ↩
    Allocate space permanently [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.SPACE.RLS="" # TYPE: NUMBER HELP: Specifies  ↩
```

```
    the type of record level sharing (RLS) being done for a specific data set [NONE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.LIKE="" # TYPE: STRING HELP: Specifies the ↩
    name of the model data set from which the attributes are to be copied #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FALLOC.RENEW="" # TYPE: SWITCH HELP: Enforce new ↩
    allocation by a remove of an existing file #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.REMAIN="" # TYPE: SWITCH HELP: Remain existing files ↩
     (don't overwrite) [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.APPEND="" # TYPE: SWITCH HELP: Append data to this ↩
    file [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.FLUSH="" # TYPE: SWITCH HELP: Enforce flush in front ↩
     of close file [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.RECMOD="" # TYPE: NUMBER HELP: Mode used to write ↩
    records [STOP] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.PRNCTR="" # TYPE: NUMBER HELP: Print control ↩
    character handling [DETACH] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.RECCNT="" # TYPE: NUMBER HELP: Initial amount of ↩
    records for preallocation [128] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT="" # TYPE: OVERLAY HELP: Format of the length ↩
     field for open platforms #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.INTEGER="" # TYPE: OBJECT HELP: Use integer ↩
    format for the length field #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.INTEGER.ENDIAN="" # TYPE: NUMBER HELP: Byte ↩
    order [SYSTEM] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.INTEGER.WIDTH="" # TYPE: NUMBER HELP: Bit ↩
    width/rate [B32] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.INTEGER.INCLUDE="" # TYPE: SWITCH HELP: ↩
    Include length field in length value [OFF] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.STRING="" # TYPE: OBJECT HELP: Use string ↩
    format for the length field #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.STRING.CHRSET="" # TYPE: NUMBER HELP: ↩
    Character set [SYSTEM] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.STRING.COUNT="" # TYPE: NUMBER HELP: ↩
    Character count [C06] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.STRING.BASE="" # TYPE: NUMBER HELP: String ↩
    base [B10] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.STRING.INCLUDE="" # TYPE: SWITCH HELP: ↩
    Include length field in length value [OFF] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.BCD="" # TYPE: OBJECT HELP: Use BCD format ↩
    for the length field #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.BCD.WIDTH="" # TYPE: NUMBER HELP: Bit width/ ↩
    rate [B32] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.BCD.INCLUDE="" # TYPE: SWITCH HELP: Include ↩
    length field in length value [OFF] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.HOST="" # TYPE: OBJECT HELP: Use host format ↩
    (LLxx) for the length field #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.HOST.ENDIAN="" # TYPE: NUMBER HELP: Byte ↩
    order [BIG ENDIAN] for first 16 Bit #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.LENFMT.HOST.EXCLUDE="" # TYPE: SWITCH HELP: Record ↩
    length field not part of the length value [OFF] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.SUPPAD="" # TYPE: SWITCH HELP: Suppress trailing ↩
    padding characters (useful for variable length records) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.PADCHR="" # TYPE: STRING HELP: Character to pad ↩
    records to a fix length #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM ↩
    ] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.RECDLM="" # TYPE: STRING HELP: Delimiter used to ↩
    define end of record #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.ACSTIM="" # TYPE: NUMBER HELP: Define the last ↩
    access time for this file #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.REC.MODTIM="" # TYPE: NUMBER HELP: Define the last ↩
    modification time for this file #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT="" # TYPE: OBJECT HELP: Write a file text-oriented ( ↩
```

```
    mainly open world) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.NAME="" # TYPE: STRING HELP: Name/URL of file to  ↩
    write [''==stdout] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC="" # TYPE: OBJECT HELP: Platform dependent  ↩
    file allocation parameter #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.ORGANIZATION="" # TYPE: NUMBER HELP: File  ↩
    organization [SEQ] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.BLKSIZE="" # TYPE: NUMBER HELP: Block size [ ↩
    system] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.RECFORMAT="" # TYPE: NUMBER HELP: Record  ↩
    format [auto] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.RECLENGTH="" # TYPE: NUMBER HELP: Record  ↩
    length (data only, no print control and length fields) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.KEYDISP="" # TYPE: NUMBER HELP: Key  ↩
    disposition for index/key sequential data sets [OLD] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.KEYPOSITION="" # TYPE: NUMBER HELP: Key  ↩
    position (>=1) for index/key sequential data sets [1] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.KEYLENGTH="" # TYPE: NUMBER HELP: Key length  ↩
    (>=1) for index/key sequential data sets [8] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SUBSYSTEM="" # TYPE: OBJECT HELP:  ↩
    Specification for FIO subsystem (only for z/OS) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP:  ↩
    Name of the subsystem #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SUBSYSTEM.PARM="'" # TYPE: STRING HELP:  ↩
    Parameter list for the subsystem (comma separated) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE="" # TYPE: OBJECT HELP: Platform  ↩
    dependent space definition parameter #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.PRIMARY="" # TYPE: NUMBER HELP: Primary ↩
     size in megabytes #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.SECONDARY="" # TYPE: NUMBER HELP:  ↩
    Secondary size in megabytes #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.DIRBLOCKS="" # TYPE: NUMBER HELP:  ↩
    Directory blocks for partitioned data sets #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.VOLSER="" # TYPE: STRING HELP: Volume  ↩
    serial number #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.VOLUNIT="" # TYPE: STRING HELP: Volume  ↩
    unit #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.DATACLASS="" # TYPE: STRING HELP: Data  ↩
    class #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.MGMTCLASS="" # TYPE: STRING HELP:  ↩
    Management class #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.STORCLASS="" # TYPE: STRING HELP:  ↩
    Storage class #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.NORMALDISP="" # TYPE: NUMBER HELP:  ↩
    Close disposition for normal end [CATALOG] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.ABORTDISP="" # TYPE: NUMBER HELP: Close ↩
     disposition for abnormal end [DELETE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.RELEASE="" # TYPE: NUMBER HELP: Release ↩
     unused space when file is closed [SEQ=YES, PDS=NO] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.UNMOVABLE="" # TYPE: SWITCH HELP:  ↩
    Allocate space unmovable [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.CONTIG="" # TYPE: SWITCH HELP: Allocate ↩
     space contiguously [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.PERMANENT="" # TYPE: SWITCH HELP:  ↩
    Allocate space permanently [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.SPACE.RLS="" # TYPE: NUMBER HELP: Specifies  ↩
    the type of record level sharing (RLS) being done for a specific data set [NONE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.LIKE="" # TYPE: STRING HELP: Specifies the  ↩
    name of the model data set from which the attributes are to be copied #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FALLOC.RENEW="" # TYPE: SWITCH HELP: Enforce new  ↩
    allocation by a remove of an existing file #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.REMAIN="" # TYPE: SWITCH HELP: Remain existing files ↩
```

```
     (don't overwrite) [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.APPEND="" # TYPE: SWITCH HELP: Append data to this ↩
    file [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.FLUSH="" # TYPE: SWITCH HELP: Enforce flush in front ↩
     of close file [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.ACSTIM="" # TYPE: NUMBER HELP: Define the last ↩
    access time for this file #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.TXT.MODTIM="" # TYPE: NUMBER HELP: Define the last ↩
    modification time for this file #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4="" # TYPE: OBJECT HELP: Write a FLAMFILE record- ↩
    oriented #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.NAME="" # TYPE: STRING HELP: Name/URL of FLAMFILE to ↩
     write #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.MEMBER="" # TYPE: STRING HELP: Rename the member in ↩
    the FLAMFILE #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.REMAIN="" # TYPE: SWITCH HELP: Remain existing files ↩
     (don't overwrite) [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.APPEND="" # TYPE: SWITCH HELP: Append the member to ↩
    this FLAMFILE (not useful with encryption) [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.CMPMOD="" # TYPE: NUMBER HELP: FLAM Mode [ADC] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.ENCMOD="" # TYPE: NUMBER HELP: Encryption Mode [NONE ↩
    ] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.BLKMOD="" # TYPE: NUMBER HELP: Block mode [unblocked ↩
    ] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.MAXBUF="" # TYPE: NUMBER HELP: Maximal buffer size ↩
    for FLAM matrix [32768] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.MAXREC="" # TYPE: NUMBER HELP: Maximal amount of ↩
    records per matrix [255] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.FLFORG="" # TYPE: NUMBER HELP: Organization of the ↩
    FLAMFILE [SEQ] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.FLFRCF="" # TYPE: NUMBER HELP: Record format for the ↩
     FLAMFILE [FIX] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.FLFRCL="" # TYPE: NUMBER HELP: Record length for the ↩
     FLAMFILE [512] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.FLFRCD="" # TYPE: NUMBER HELP: Record delimiter for ↩
    the FLAMFILE (only for CX7) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.FLFBLS="" # TYPE: NUMBER HELP: Block size for the ↩
    FLAMFILE (mainly host related) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.FLFCDP="" # TYPE: NUMBER HELP: Close disposition for ↩
     the FLAMFILE (mainly host related) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.FLFCOD="" # TYPE: NUMBER HELP: Character set for ↩
    string encoding in the FLAMFILE [SYSTEM] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.SPLMOD="" # TYPE: NUMBER HELP: Split mode [NONE] ( ↩
    only for z/OS) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.SPLNBR="" # TYPE: NUMBER HELP: Split number (2-4 ↩
    files for parallel splitting (only for z/OS)) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.SPLSIZ="" # TYPE: NUMBER HELP: Split size (1-4096 MB ↩
     for serial splitting (only for z/OS)) #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.ORGORG="" # TYPE: NUMBER HELP: Organization of the ↩
    original file [from origin] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.ORGRCF="" # TYPE: NUMBER HELP: Record format for the ↩
     original file [from origin] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.ORGRCL="" # TYPE: NUMBER HELP: Record length for the ↩
     original file [from origin] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.ORGBLS="" # TYPE: NUMBER HELP: Block size for the ↩
    original file [from origin] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.ORGPRN="" # TYPE: NUMBER HELP: Print control byte ↩
    for ASA or machine control character [from origin] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.ORGRCD="" # TYPE: NUMBER HELP: Record delimiter for ↩
    the original file [from origin] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.ORGSYS="" # TYPE: STRING HELP: Operating system for ↩
    the original file [from origin] #
```

```
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.DEVICE="" # TYPE: NUMBER HELP: Device used to store ↩
    the FLAMFILE [FLATFILE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.NONHDR="" # TYPE: SWITCH HELP: Switch to write no ↩
    file header [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.NONFIL="" # TYPE: SWITCH HELP: Switch to write no ↩
    file name in file header [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.COMENT="" # TYPE: STRING HELP: Comment for the ↩
    member in the FLAMFILE [internal use] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.PASSWD="" # TYPE: STRING HELP: Passphrase to encrypt ↩
     the FLAMFILE [’’] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.KMELIB="" # TYPE: STRING HELP: Library name for FKME ↩
     [’’] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.KMEFUC="" # TYPE: STRING HELP: Function name of FKME ↩
     [’’] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.KMEPAR="" # TYPE: STRING HELP: Parameter for the ↩
    call of FKME [’’] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.EXTCMP="" # TYPE: STRING HELP: Name of user exit for ↩
     compression #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.EXTDCO="" # TYPE: STRING HELP: Name of user exit for ↩
     decompression #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.ACSTIM="" # TYPE: NUMBER HELP: Define the last ↩
    access time for this file #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.MODTIM="" # TYPE: NUMBER HELP: Define the last ↩
    modification time for this file #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.BLKORI="" # TYPE: SWITCH HELP: Enforce block- ↩
    oriented write of records (32752) with FLAM4 #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.PRISPC="" # TYPE: NUMBER HELP: Primary size in ↩
    megabytes #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.SECSPC="" # TYPE: NUMBER HELP: Secondary size in ↩
    megabytes #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.VOLSER="" # TYPE: STRING HELP: Volume serial number ↩
    #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.VOLUNT="" # TYPE: STRING HELP: Volume unit #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.DATCLS="" # TYPE: STRING HELP: Data class #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.MGMCLS="" # TYPE: STRING HELP: Management class #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.STOCLS="" # TYPE: STRING HELP: Storage class #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.DSPSTA="" # TYPE: NUMBER HELP: Disposition status #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.DSPNRM="" # TYPE: NUMBER HELP: Close disposition for ↩
     normal end #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.DSPABO="" # TYPE: NUMBER HELP: Close disposition for ↩
     abnormal end #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.KEYPOS="" # TYPE: NUMBER HELP: Key position (>=1) ↩
    for index/key sequential data sets [1] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.KEYLEN="" # TYPE: NUMBER HELP: Key length (>=1) for ↩
    index/key sequential data sets [8] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.SECINF="" # TYPE: NUMBER HELP: Define handling of ↩
    secure information #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.FIO.FL4.NOPATH="" # TYPE: SWITCH HELP: Exclude directory ↩
    path of the original file in the compressed data [FALSE] #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.ENV="" # TYPE: OBJECT HELP: Environment #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.ENV.KEYWORD="" # TYPE: STRING HELP: Keyword of environment ↩
    variable #
LIM.flcl.XCNV.OUTPUT.SAV.FILE.ENV.VALUE="" # TYPE: STRING HELP: Value of environment ↩
    variable #
LIM.flcl.XCNV.DIR="" # TYPE: OBJECT HELP: Defines how to handle directories and various ↩
    file types if wildcards used #
LIM.flcl.XCNV.DIR.LINK="" # TYPE: SWITCH HELP: Include symbolic links #
LIM.flcl.XCNV.DIR.ALIAS="" # TYPE: SWITCH HELP: Resolve data set aliases #
LIM.flcl.XCNV.DIR.HIDDEN="" # TYPE: SWITCH HELP: Include hidden files #
LIM.flcl.XCNV.DIR.RECURSIVE="" # TYPE: SWITCH HELP: Include sub directories #
LIM.flcl.XCNV.DIR.ARCHIVE="" # TYPE: SWITCH HELP: Include archives (FLAMFILEs, ZIP, ...) #
LIM.flcl.XCNV.DIR.ONEFILESYSTEM="" # TYPE: SWITCH HELP: Don’t cross file system boundaries ↩
```

```
     #
LIM.flcl.XCNV.DIR.TAPE="" # TYPE: SWITCH HELP: Include files deposited on tapes #
LIM.flcl.XCNV.LOG="" # TYPE: OBJECT HELP: Logging Parameter #
LIM.flcl.XCNV.LOG.MESSAGE="" # TYPE: OBJECT HELP: Select type of messages to log #
LIM.flcl.XCNV.LOG.MESSAGE.NONE="" # TYPE: SWITCH HELP: No messages will be logged #
LIM.flcl.XCNV.LOG.MESSAGE.ERROR="" # TYPE: SWITCH HELP: Error messages will be logged #
LIM.flcl.XCNV.LOG.MESSAGE.ERRTRACE="" # TYPE: SWITCH HELP: Error trace will be logged #
LIM.flcl.XCNV.LOG.MESSAGE.WARNING="" # TYPE: SWITCH HELP: Warnings will be logged #
LIM.flcl.XCNV.LOG.MESSAGE.NOTICE="" # TYPE: SWITCH HELP: Notices will be logged #
LIM.flcl.XCNV.LOG.MESSAGE.DEBUG="" # TYPE: SWITCH HELP: Debug information will be logged #
LIM.flcl.XCNV.LOG.MESSAGE.ABOUT="" # TYPE: SWITCH HELP: About information will be logged #
LIM.flcl.XCNV.LOG.MESSAGE.VERSION="" # TYPE: SWITCH HELP: Version information will be ↩
    logged #
LIM.flcl.XCNV.LOG.MESSAGE.INPUT="" # TYPE: SWITCH HELP: Input data will be logged ( ↩
    dangerous for passwords) #
LIM.flcl.XCNV.LOG.MESSAGE.PARAMETER="" # TYPE: SWITCH HELP: Parsed parameters will be ↩
    logged #
LIM.flcl.XCNV.LOG.MESSAGE.STATISTIC="" # TYPE: SWITCH HELP: Statistics will be logged #
LIM.flcl.XCNV.LOG.MESSAGE.INFO="" # TYPE: SWITCH HELP: Requested information will be ↩
    logged #
LIM.flcl.XCNV.LOG.MESSAGE.LICID="" # TYPE: SWITCH HELP: License ID will be logged #
LIM.flcl.XCNV.LOG.MESSAGE.LICENSE="" # TYPE: SWITCH HELP: License summary will be logged #
LIM.flcl.XCNV.LOG.MESSAGE.PROGRESS="" # TYPE: SWITCH HELP: Progress bar will be logged #
LIM.flcl.XCNV.LOG.MESSAGE.ALL="" # TYPE: SWITCH HELP: All messages are logged #
LIM.flcl.XCNV.LOG.MESSAGE.DEFAULT="" # TYPE: SWITCH HELP: Relevant messages are logged #
LIM.flcl.XCNV.LOG.MESSAGE.MINIMAL="" # TYPE: SWITCH HELP: Only important messages are ↩
    logged #
LIM.flcl.XCNV.LOG.MESSAGE.ERRONLY="" # TYPE: SWITCH HELP: Only errors and error trace are ↩
    logged #
LIM.flcl.XCNV.LOG.DESTINATION="" # TYPE: OVERLAY HELP: Destination for log messages #
LIM.flcl.XCNV.LOG.DESTINATION.STREAM="" # TYPE: OBJECT HELP: Log to a stream #
LIM.flcl.XCNV.LOG.DESTINATION.STREAM.FORMAT="" # TYPE: NUMBER HELP: Format of log message ↩
    [STANDARD] #
LIM.flcl.XCNV.LOG.DESTINATION.STREAM.IDENT="" # TYPE: STRING HELP: Identifier used to ↩
    identify log entries #
LIM.flcl.XCNV.LOG.DESTINATION.STREAM.STDOUT="" # TYPE: SWITCH HELP: Log messages to STDOUT ↩
     instead of STDERR #
LIM.flcl.XCNV.LOG.DESTINATION.FILE="" # TYPE: OBJECT HELP: Log to a flat file #
LIM.flcl.XCNV.LOG.DESTINATION.FILE.FORMAT="" # TYPE: NUMBER HELP: Format of log message [ ↩
    STANDARD] #
LIM.flcl.XCNV.LOG.DESTINATION.FILE.IDENT="" # TYPE: STRING HELP: Identifier used to ↩
    identify log entries #
LIM.flcl.XCNV.LOG.DESTINATION.FILE.NAME="" # TYPE: STRING HELP: Filename for appending log ↩
     messages #
LIM.flcl.XCNV.LOG.DESTINATION.SYSTEM="" # TYPE: OBJECT HELP: Use system logging facility #
LIM.flcl.XCNV.LOG.DESTINATION.SYSTEM.IDENT="" # TYPE: STRING HELP: Identifier used to ↩
    identify log entries #
LIM.flcl.ICNV.IN="" # TYPE: STRING HELP: Input file/URL [required] #
LIM.flcl.ICNV.OUT="" # TYPE: STRING HELP: Output file/URL [required] #
LIM.flcl.ICNV.FALLOC="" # TYPE: OBJECT HELP: Platform dependent file allocation parameter ↩
    for output #
LIM.flcl.ICNV.FALLOC.ORGANIZATION="" # TYPE: NUMBER HELP: File organization [SEQ] #
LIM.flcl.ICNV.FALLOC.BLKSIZE="" # TYPE: NUMBER HELP: Block size [system] #
LIM.flcl.ICNV.FALLOC.RECFORMAT="" # TYPE: NUMBER HELP: Record format [auto] #
LIM.flcl.ICNV.FALLOC.RECLENGTH="" # TYPE: NUMBER HELP: Record length (data only, no print ↩
    control and length fields) #
LIM.flcl.ICNV.FALLOC.KEYDISP="" # TYPE: NUMBER HELP: Key disposition for index/key ↩
    sequential data sets [OLD] #
LIM.flcl.ICNV.FALLOC.KEYPOSITION="" # TYPE: NUMBER HELP: Key position (>=1) for index/key ↩
    sequential data sets [1] #
LIM.flcl.ICNV.FALLOC.KEYLENGTH="" # TYPE: NUMBER HELP: Key length (>=1) for index/key ↩
    sequential data sets [8] #
```

```
LIM.flcl.ICNV.FALLOC.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO subsystem ( ↩
    only for z/OS) #
LIM.flcl.ICNV.FALLOC.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the subsystem #
LIM.flcl.ICNV.FALLOC.SUBSYSTEM.PARM="'" # TYPE: STRING HELP: Parameter list for the  ↩
    subsystem (comma separated) #
LIM.flcl.ICNV.FALLOC.SPACE="" # TYPE: OBJECT HELP: Platform dependent space definition  ↩
    parameter #
LIM.flcl.ICNV.FALLOC.SPACE.PRIMARY="" # TYPE: NUMBER HELP: Primary size in megabytes #
LIM.flcl.ICNV.FALLOC.SPACE.SECONDARY="" # TYPE: NUMBER HELP: Secondary size in megabytes #
LIM.flcl.ICNV.FALLOC.SPACE.DIRBLOCKS="" # TYPE: NUMBER HELP: Directory blocks for  ↩
    partitioned data sets #
LIM.flcl.ICNV.FALLOC.SPACE.VOLSER="" # TYPE: STRING HELP: Volume serial number #
LIM.flcl.ICNV.FALLOC.SPACE.VOLUNIT="" # TYPE: STRING HELP: Volume unit #
LIM.flcl.ICNV.FALLOC.SPACE.DATACLASS="" # TYPE: STRING HELP: Data class #
LIM.flcl.ICNV.FALLOC.SPACE.MGMTCLASS="" # TYPE: STRING HELP: Management class #
LIM.flcl.ICNV.FALLOC.SPACE.STORCLASS="" # TYPE: STRING HELP: Storage class #
LIM.flcl.ICNV.FALLOC.SPACE.NORMALDISP="" # TYPE: NUMBER HELP: Close disposition for normal ↩
     end [CATALOG] #
LIM.flcl.ICNV.FALLOC.SPACE.ABORTDISP="" # TYPE: NUMBER HELP: Close disposition for  ↩
    abnormal end [DELETE] #
LIM.flcl.ICNV.FALLOC.SPACE.RELEASE="" # TYPE: NUMBER HELP: Release unused space when file  ↩
    is closed [SEQ=YES, PDS=NO] #
LIM.flcl.ICNV.FALLOC.SPACE.UNMOVABLE="" # TYPE: SWITCH HELP: Allocate space unmovable [ ↩
    FALSE] #
LIM.flcl.ICNV.FALLOC.SPACE.CONTIG="" # TYPE: SWITCH HELP: Allocate space contiguously [ ↩
    FALSE] #
LIM.flcl.ICNV.FALLOC.SPACE.PERMANENT="" # TYPE: SWITCH HELP: Allocate space permanently [ ↩
    FALSE] #
LIM.flcl.ICNV.FALLOC.SPACE.RLS="" # TYPE: NUMBER HELP: Specifies the type of record level  ↩
    sharing (RLS) being done for a specific data set [NONE] #
LIM.flcl.ICNV.FALLOC.LIKE="" # TYPE: STRING HELP: Specifies the name of the model data set ↩
     from which the attributes are to be copied #
LIM.flcl.ICNV.FALLOC.RENEW="" # TYPE: SWITCH HELP: Enforce new allocation by a remove of  ↩
    an existing file #
LIM.flcl.ICNV.RECORD="" # TYPE: SWITCH HELP: Use record-oriented file IO instead of block- ↩
    oriented IO #
LIM.flcl.ICNV.METHOD="" # TYPE: NUMBER HELP: Enforces record or block-oriented conversion  ↩
    [AUTO] #
LIM.flcl.ICNV.CASE="" # TYPE: NUMBER HELP: Define upper, lower or special case conversion  ↩
    [NONE] #
LIM.flcl.ICNV.FROM="" # TYPE: STRING HELP: Conversion from this CCSID [auto detection] #
LIM.flcl.ICNV.MODE="" # TYPE: NUMBER HELP: Define handling of non-convertible characters #
LIM.flcl.ICNV.BOM="" # TYPE: SWITCH HELP: Write byte order mark in front of the output  ↩
    text #
LIM.flcl.ICNV.TO="" # TYPE: STRING HELP: Conversion to this CCSID [UTF-8] #
LIM.flcl.ICNV.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15) by line feed  ↩
    (0x25) on input #
LIM.flcl.ICNV.ELF2NL="" # TYPE: SWITCH HELP: Replace EBCDIC line feed (0x25) by new line  ↩
    (0x15) on output #
LIM.flcl.ICNV.SUBCHR="" # TYPE: NUMBER HELP: Default substitution character list [0x1A] ( ↩
    maximal 8 code points)  #
LIM.flcl.ICNV.SYSTAB="" # TYPE: NUMBER HELP: Define a preloaded system substitution table  ↩
    #
LIM.flcl.ICNV.USRTAB="" # TYPE: STRING HELP: Name of the file containing the user  ↩
    conversion table #
LIM.flcl.ICNV.REPORT="" # TYPE: STRING HELP: Name of the report file to log substitutions  ↩
    #
LIM.flcl.ICNV.REMAIN="" # TYPE: SWITCH HELP: Remain existing files (don't overwrite) [ ↩
    FALSE] #
LIM.flcl.ICNV.APPEND="" # TYPE: SWITCH HELP: Append data to output file [FALSE] #
LIM.flcl.ICNV.REMOVE="" # TYPE: SWITCH HELP: Remove original file after successful  ↩
    processing [FALSE] #
```

```
 LIM.flcl.ICNV.DIR="" # TYPE: OBJECT HELP: Defines how to handle directories and various  ↩
     file types if wildcards used #
 LIM.flcl.ICNV.DIR.LINK="" # TYPE: SWITCH HELP: Include symbolic links #
 LIM.flcl.ICNV.DIR.ALIAS="" # TYPE: SWITCH HELP: Resolve data set aliases #
 LIM.flcl.ICNV.DIR.HIDDEN="" # TYPE: SWITCH HELP: Include hidden files #
 LIM.flcl.ICNV.DIR.RECURSIVE="" # TYPE: SWITCH HELP: Include sub directories #
 LIM.flcl.ICNV.DIR.ARCHIVE="" # TYPE: SWITCH HELP: Include archives (FLAMFILEs, ZIP, ...) #
 LIM.flcl.ICNV.DIR.ONEFILESYSTEM="" # TYPE: SWITCH HELP: Don't cross file system boundaries ↩
      #
 LIM.flcl.ICNV.DIR.TAPE="" # TYPE: SWITCH HELP: Include files deposited on tapes #
 LIM.flcl.ICNV.LOG="" # TYPE: OBJECT HELP: Logging Parameter #
 LIM.flcl.ICNV.LOG.MESSAGE="" # TYPE: OBJECT HELP: Select type of messages to log #
 LIM.flcl.ICNV.LOG.MESSAGE.NONE="" # TYPE: SWITCH HELP: No messages will be logged #
 LIM.flcl.ICNV.LOG.MESSAGE.ERROR="" # TYPE: SWITCH HELP: Error messages will be logged #
 LIM.flcl.ICNV.LOG.MESSAGE.ERRTRACE="" # TYPE: SWITCH HELP: Error trace will be logged #
 LIM.flcl.ICNV.LOG.MESSAGE.WARNING="" # TYPE: SWITCH HELP: Warnings will be logged #
 LIM.flcl.ICNV.LOG.MESSAGE.NOTICE="" # TYPE: SWITCH HELP: Notices will be logged #
 LIM.flcl.ICNV.LOG.MESSAGE.DEBUG="" # TYPE: SWITCH HELP: Debug information will be logged #
 LIM.flcl.ICNV.LOG.MESSAGE.ABOUT="" # TYPE: SWITCH HELP: About information will be logged #
 LIM.flcl.ICNV.LOG.MESSAGE.VERSION="" # TYPE: SWITCH HELP: Version information will be  ↩
     logged #
 LIM.flcl.ICNV.LOG.MESSAGE.INPUT="" # TYPE: SWITCH HELP: Input data will be logged ( ↩
     dangerous for passwords) #
 LIM.flcl.ICNV.LOG.MESSAGE.PARAMETER="" # TYPE: SWITCH HELP: Parsed parameters will be  ↩
     logged #
 LIM.flcl.ICNV.LOG.MESSAGE.STATISTIC="" # TYPE: SWITCH HELP: Statistics will be logged #
 LIM.flcl.ICNV.LOG.MESSAGE.INFO="" # TYPE: SWITCH HELP: Requested information will be  ↩
     logged #
 LIM.flcl.ICNV.LOG.MESSAGE.LICID="" # TYPE: SWITCH HELP: License ID will be logged #
 LIM.flcl.ICNV.LOG.MESSAGE.LICENSE="" # TYPE: SWITCH HELP: License summary will be logged #
 LIM.flcl.ICNV.LOG.MESSAGE.PROGRESS="" # TYPE: SWITCH HELP: Progress bar will be logged #
 LIM.flcl.ICNV.LOG.MESSAGE.ALL="" # TYPE: SWITCH HELP: All messages are logged #
 LIM.flcl.ICNV.LOG.MESSAGE.DEFAULT="" # TYPE: SWITCH HELP: Relevant messages are logged #
 LIM.flcl.ICNV.LOG.MESSAGE.MINIMAL="" # TYPE: SWITCH HELP: Only important messages are  ↩
     logged #
 LIM.flcl.ICNV.LOG.MESSAGE.ERRONLY="" # TYPE: SWITCH HELP: Only errors and error trace are  ↩
     logged #
 LIM.flcl.ICNV.LOG.DESTINATION="" # TYPE: OVERLAY HELP: Destination for log messages #
 LIM.flcl.ICNV.LOG.DESTINATION.STREAM="" # TYPE: OBJECT HELP: Log to a stream #
 LIM.flcl.ICNV.LOG.DESTINATION.STREAM.FORMAT="" # TYPE: NUMBER HELP: Format of log message  ↩
     [STANDARD] #
 LIM.flcl.ICNV.LOG.DESTINATION.STREAM.IDENT="" # TYPE: STRING HELP: Identifier used to  ↩
     identify log entries #
 LIM.flcl.ICNV.LOG.DESTINATION.STREAM.STDOUT="" # TYPE: SWITCH HELP: Log messages to STDOUT ↩
      instead of STDERR #
 LIM.flcl.ICNV.LOG.DESTINATION.FILE="" # TYPE: OBJECT HELP: Log to a flat file #
 LIM.flcl.ICNV.LOG.DESTINATION.FILE.FORMAT="" # TYPE: NUMBER HELP: Format of log message [ ↩
     STANDARD] #
 LIM.flcl.ICNV.LOG.DESTINATION.FILE.IDENT="" # TYPE: STRING HELP: Identifier used to  ↩
     identify log entries #
 LIM.flcl.ICNV.LOG.DESTINATION.FILE.NAME="" # TYPE: STRING HELP: Filename for appending log ↩
      messages #
 LIM.flcl.ICNV.LOG.DESTINATION.SYSTEM="" # TYPE: OBJECT HELP: Use system logging facility #
 LIM.flcl.ICNV.LOG.DESTINATION.SYSTEM.IDENT="" # TYPE: STRING HELP: Identifier used to  ↩
     identify log entries #

# DESCRIPTION for LIM.flcl.FLAM.NETINP:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.
```

```
 #
 LIM.flcl.FLAM.NETINP="" # TYPE: OVERLAY HELP: Network parameter for remote read access #

# DESCRIPTION for LIM.flcl.FLAM.NETINP.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.FLAM.NETINP.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.FLAM.NETINP.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.FLAM.NETINP.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.FLAM.NETINP.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.FLAM.NETINP.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #

# DESCRIPTION for LIM.flcl.FLAM.NETINP.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.FLAM.NETINP.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.FLAM.NETINP.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.FLAM.NETINP.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.FLAM.NETINP.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.FLAM.NETINP.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #

# DESCRIPTION for LIM.flcl.FLAM.NETINP.MQS:
```

```
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.FLAM.NETINP.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.FLAM.NETINP.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
 LIM.flcl.FLAM.NETINP.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.FLAM.NETINP.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.FLAM.NETINP.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #

# DESCRIPTION for LIM.flcl.FLAM.NETOUT:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.FLAM.NETOUT="" # TYPE: OVERLAY HELP: Network parameter for remote write access #

# DESCRIPTION for LIM.flcl.FLAM.NETOUT.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.FLAM.NETOUT.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.FLAM.NETOUT.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.FLAM.NETOUT.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.FLAM.NETOUT.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.FLAM.NETOUT.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #

# DESCRIPTION for LIM.flcl.FLAM.NETOUT.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
```

```
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.FLAM.NETOUT.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.FLAM.NETOUT.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.FLAM.NETOUT.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.FLAM.NETOUT.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.FLAM.NETOUT.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #

# DESCRIPTION for LIM.flcl.FLAM.NETOUT.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.FLAM.NETOUT.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.FLAM.NETOUT.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
 LIM.flcl.FLAM.NETOUT.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.FLAM.NETOUT.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.FLAM.NETOUT.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #

 LIM.flcl.FLAM.COMPRESS="" # TYPE: SWITCH HELP: Activate compression (write) [FALSE] #
 LIM.flcl.FLAM.ADD="" # TYPE: SWITCH HELP: Compressed data will be added to a FLAMFILE #
 LIM.flcl.FLAM.DECOMPRESS="" # TYPE: SWITCH HELP: Activate decompression (read) [TRUE] #
 LIM.flcl.FLAM.REMAIN="" # TYPE: SWITCH HELP: Remain existing files (don't overwrite) [ ↩
     FALSE] #
 LIM.flcl.FLAM.APPEND="" # TYPE: SWITCH HELP: The decompressed data is appended to a file #
 LIM.flcl.FLAM.FLAMFILE="" # TYPE: STRING HELP: Name/URL of FLAMFILE for read or write  ↩
     operation #
 LIM.flcl.FLAM.FLAMIN="" # TYPE: STRING HELP: Name/URL of the files to be compressed #
 LIM.flcl.FLAM.FLAMOUT="" # TYPE: STRING HELP: Name/URL of the files to be decompressed #
 LIM.flcl.FLAM.MODE="" # TYPE: NUMBER HELP: Mode in which the original file is compressed  ↩
     and, when desired, encrypted #
 LIM.flcl.FLAM.CRYPTMODE="" # TYPE: NUMBER HELP: Encryption mode (Only with mode=adc or  ↩
     mode=ndc) [NONE] #
 LIM.flcl.FLAM.CRYPTKEY="" # TYPE: STRING HELP: Password for en/de-crypting a FLAMFILE in  ↩
     AES or FLAMENC mode (default: AES). #
```

```
LIM.flcl.FLAM.PASSWORD="" # TYPE: STRING HELP: Password for en/de-crypting a FLAMFILE in AES or FLAMENC mode (default: FLAMENC). #
LIM.flcl.FLAM.PAASCII="" # TYPE: STRING HELP: Password in ASCII for en/de-crypting in AES or FLAMENC mode (default: FLAMENC). #
LIM.flcl.FLAM.PAEBCDIC="" # TYPE: STRING HELP: Password in EBCIDC for en/de-crypting in AES or FLAMENC mode (default: FLAMENC). #
LIM.flcl.FLAM.PAXWORD="" # TYPE: STRING HELP: Password in HEX for en/de-crypting in AES or FLAMENC mode (default: FLAMENC). #
LIM.flcl.FLAM.PAFILE="" # TYPE: STRING HELP: File with parameters for en- and decryption (other are also permitted)[''] #
LIM.flcl.FLAM.RECDELIM="" # TYPE: STRING HELP: Record length of original files (format "fix"/"undefined", 1-32760) [512] #
LIM.flcl.FLAM.RECFORMAT="" # TYPE: NUMBER HELP: Record format of FLAMFILE [FIX] #
LIM.flcl.FLAM.RECSIZE="" # TYPE: NUMBER HELP: Record length of FLAMFILE (modes CX8,VR8,ADC,NDC,AES 80-32760)(mode CX7 80-4095) [512] #
LIM.flcl.FLAM.INRECDELIM="" # TYPE: STRING HELP: Record delimiter for original files #
LIM.flcl.FLAM.INRECFORMAT="" # TYPE: NUMBER HELP: Record format of the original file #
LIM.flcl.FLAM.INRECSIZE="" # TYPE: NUMBER HELP: Record length of original files (format "fix"/"undefined", 1-32760) [512] #
LIM.flcl.FLAM.OUTRECDELIM="" # TYPE: STRING HELP: Record delimiter for decompressed files #
LIM.flcl.FLAM.OUTRECFORMAT="" # TYPE: NUMBER HELP: Record format of decompressed file #
LIM.flcl.FLAM.OUTRECSIZE="" # TYPE: NUMBER HELP: Record length of decompressed files ( format "fix"/"undefined", 1-32760) [512] #
LIM.flcl.FLAM.BLKSIZE="" # TYPE: NUMBER HELP: Logical block length for compressed file 0 - 32760 [0] #
LIM.flcl.FLAM.IBLKSIZE="" # TYPE: NUMBER HELP: Block length of the input file 0 - 32760 [0] #
LIM.flcl.FLAM.OBLKSIZE="" # TYPE: NUMBER HELP: Block length of the output file 0 - 32760 [0] #
LIM.flcl.FLAM.ATTRIBUTES="" # TYPE: NUMBER HELP: Causes that original file information to be entered in the FLAMFILE [COMMON] #
LIM.flcl.FLAM.OPTION="" # TYPE: NUMBER HELP: Controls FLAM's behavior when decompressed records are output #
LIM.flcl.FLAM.CUT="" # TYPE: SWITCH HELP: Records with a record length longer than the maximum record length may be shortened #
LIM.flcl.FLAM.NOCUT="" # TYPE: SWITCH HELP: Records with a record length larger than the maximum record length are not cut #
LIM.flcl.FLAM.SUPPRESS="" # TYPE: SWITCH HELP: Blanks at the end of record are suppressed #
LIM.flcl.FLAM.NOSUPPRESS="" # TYPE: SWITCH HELP: Blanks at the end of record are not suppressed #
LIM.flcl.FLAM.PAD_CHAR="" # TYPE: STRING HELP: Character to pad records to a fix length #
LIM.flcl.FLAM.MAXBUFFER="" # TYPE: NUMBER HELP: Maximal buffer size for FLAM matrix [32768] #
LIM.flcl.FLAM.MAXRECORDS="" # TYPE: NUMBER HELP: Maximal amount of records per matrix [255] #
LIM.flcl.FLAM.FLAMCODE="" # TYPE: NUMBER HELP: Character set for string encoding in the FLAMFILE [SYSTEM] #
LIM.flcl.FLAM.KMELIB="" # TYPE: STRING HELP: Library name for FKME [''] #
LIM.flcl.FLAM.KMEXIT="" # TYPE: STRING HELP: Function name of FKME [''] #
LIM.flcl.FLAM.KMPARAM="" # TYPE: STRING HELP: Parameter for the call of FKME [''] #
LIM.flcl.FLAM.HEADER="" # TYPE: NUMBER HELP: Create file header [YES] #
LIM.flcl.FLAM.COMMENT="" # TYPE: STRING HELP: Comment for the member in the FLAMFILE [ internal use] #
LIM.flcl.FLAM.MSGFILE="" # TYPE: STRING HELP: Filename for appending FLAM messages #
LIM.flcl.FLAM.SECUREINFO="" # TYPE: NUMBER HELP: Define handling of secure information #
LIM.flcl.FLAM.MSGDISP="" # TYPE: NUMBER HELP: Output medium for FLAM messages #
LIM.flcl.FLAM.SHOW="" # TYPE: NUMBER HELP: Specifies which information should be shown #
LIM.flcl.FLAM.FILEINFO="" # TYPE: NUMBER HELP: Transfer file name of original into file header [YES] #
LIM.flcl.FLAM.CODETABLE="" # TYPE: STRING HELP: Code table file name #
```

```
LIM.flcl.FLAM.TRANSLATE="" # TYPE: STRING HELP: Turns translation of original data on #
LIM.flcl.FLAM.PARFILE="" # TYPE: STRING HELP: Parameter file, overrule command line ↩
    parameter [''] #
LIM.flcl.FLAM.NOPATH="" # TYPE: SWITCH HELP: Exclude directory path of the original file ↩
    in the compressed data [FALSE] #
LIM.flcl.FLAM.CLIMIT="" # TYPE: NUMBER HELP: Minimal compression in per cent (0 no Limit) ↩
    [0] #
LIM.flcl.FLAM.IPRCNTRL="" # TYPE: NUMBER HELP: Print control character handling if read ↩
    ASA or MCC records [DETACH] #
LIM.flcl.FLAM.OPRCNTRL="" # TYPE: NUMBER HELP: Print control character handling if write ↩
    ASA or MCC records [DETACH] #
LIM.flcl.FLAM.SPACE1="" # TYPE: NUMBER HELP: (HOST) Primary space allocation of the ↩
    FLAMFILE in Megabytes [2] #
LIM.flcl.FLAM.SPACE2="" # TYPE: NUMBER HELP: (HOST) Secondary space allocation of the ↩
    FLAMFILE in Megabytes [40] #
LIM.flcl.FLAM.OSPACE1="" # TYPE: NUMBER HELP: (HOST) Primary space allocation of the ↩
    output file in Megabytes [4] #
LIM.flcl.FLAM.OSPACE2="" # TYPE: NUMBER HELP: (HOST) Secondary space allocation of the ↩
    output file in Megabytes [50] #
LIM.flcl.FLAM.UNIT="" # TYPE: STRING HELP: (HOST) Unit for allocation of the FLAMFILE #
LIM.flcl.FLAM.OUNIT="" # TYPE: STRING HELP: (HOST) Unit for allocation of the output file ↩
    #
LIM.flcl.FLAM.STORCLASS="" # TYPE: STRING HELP: (HOST) Storage class  for allocation of ↩
    the FLAMFILE #
LIM.flcl.FLAM.OSTORCLASS="" # TYPE: STRING HELP: (HOST) Storage class  for allocation of ↩
    the output file #
LIM.flcl.FLAM.VOLUME="" # TYPE: STRING HELP: (HOST) Volume for allocation of the output ↩
    file #
LIM.flcl.FLAM.OVOLUME="" # TYPE: STRING HELP: (HOST) Volume for allocation of the output ↩
    file #
LIM.flcl.FLAM.DATACLASS="" # TYPE: STRING HELP: (HOST) Data storage class for allocation ↩
    of the FLAMFILE #
LIM.flcl.FLAM.ODATACLASS="" # TYPE: STRING HELP: (HOST) Data storage class for output file ↩
     #
LIM.flcl.FLAM.MGMTCLASS="" # TYPE: STRING HELP: (HOST) SMS management class for allocation ↩
     of the FLAMFILE #
LIM.flcl.FLAM.OMGMTCLASS="" # TYPE: STRING HELP: (HOST) Management class for allocation of ↩
     the output file #
LIM.flcl.FLAM.DSORG="" # TYPE: NUMBER HELP: (HOST) File organization for compressed file #
LIM.flcl.FLAM.IDSORG="" # TYPE: NUMBER HELP: (HOST) File organization for input file #
LIM.flcl.FLAM.ODSORG="" # TYPE: NUMBER HELP: (HOST) File organization for output file #
LIM.flcl.FLAM.CLOSDISP="" # TYPE: NUMBER HELP: (HOST) Close disposition for the FLAMFILE  ↩
    #
LIM.flcl.FLAM.DISPS="" # TYPE: NUMBER HELP: (HOST) DISP-Status of the FLAMFILE #
LIM.flcl.FLAM.ODISPS="" # TYPE: NUMBER HELP: (HOST) DISP-Status of the FLAMOUT file #
LIM.flcl.FLAM.DISPNORMAL="" # TYPE: NUMBER HELP: (HOST) Close disposition for normal end [ ↩
    CATALOG] #
LIM.flcl.FLAM.ODISPNORMAL="" # TYPE: NUMBER HELP: (HOST) Normal disposition of the FLAMOUT ↩
     file #
LIM.flcl.FLAM.DISPABNORMAL="" # TYPE: NUMBER HELP: (HOST) Close disposition for abnormal ↩
    end [DELETE] #
LIM.flcl.FLAM.ODISPABNORMAL="" # TYPE: NUMBER HELP: (HOST) Abnormal disposition of the ↩
    FLAMOUT file [DELETE] #
LIM.flcl.FLAM.KEYDISP="" # TYPE: NUMBER HELP: (HOST) Key disposition for index/key ↩
    sequential data sets [OLD] #
LIM.flcl.FLAM.IKEYPOSITION="" # TYPE: NUMBER HELP: (HOST) Key position of input file [1] #
LIM.flcl.FLAM.OKEYPOSITION="" # TYPE: NUMBER HELP: (HOST) Key position of output file [1] ↩
    #
LIM.flcl.FLAM.IKEYLENGTH="" # TYPE: NUMBER HELP: (HOST) Key length of input file  #
LIM.flcl.FLAM.OKEYLENGTH="" # TYPE: NUMBER HELP: (HOST) Key length of output original file ↩
     [8] #
LIM.flcl.FLAM.DEVICE="" # TYPE: NUMBER HELP: (HOST) Device used to store the FLAMFILE [ ↩
```

```
    FLATFILE] #
LIM.flcl.FLAM.IDEVICE="" # TYPE: NUMBER HELP: (HOST) Device used to store the FLAMFILE [ ↩
    FLATFILE] #
LIM.flcl.FLAM.ODEVICE="" # TYPE: NUMBER HELP: (HOST) Device used to store the FLAMFILE [ ↩
    FLATFILE] #
LIM.flcl.FLAM.SPLITMODE="" # TYPE: NUMBER HELP: (HOST) Split mode [NONE] #
LIM.flcl.FLAM.SPLITNUMBER="" # TYPE: NUMBER HELP: (HOST) Split number (2-4 files for  ↩
    parallel splitting) #
LIM.flcl.FLAM.SPLITSIZE="" # TYPE: NUMBER HELP: (HOST) Split size (1-4096 MB for serial  ↩
    splitting) #
LIM.flcl.FLAM.FLAMDDN="" # TYPE: STRING HELP: (HOST) Symbolic file name for compressed  ↩
    file #
LIM.flcl.FLAM.IDDN="" # TYPE: STRING HELP: (HOST) Symbolic file name for input file #
LIM.flcl.FLAM.ODDN="" # TYPE: STRING HELP: (HOST) Symbolic file name for output file #
LIM.flcl.FLAM.MSGDDN="" # TYPE: STRING HELP: (HOST) DD name for a message file #
LIM.flcl.FLAM.FROM="" # TYPE: STRING HELP: Conversion from this CCSID #
LIM.flcl.FLAM.SUBMODE="" # TYPE: NUMBER HELP: Define handling of non-convertible  ↩
    characters #
LIM.flcl.FLAM.BOM="" # TYPE: SWITCH HELP: Write byte order mark in front of the output  ↩
    text #
LIM.flcl.FLAM.TO="" # TYPE: STRING HELP: Conversion to this CCSID [UTF-8] #
LIM.flcl.FLAM.DIR="" # TYPE: OBJECT HELP: Defines how to handle directories and various  ↩
    file types if wildcards used #
LIM.flcl.FLAM.DIR.LINK="" # TYPE: SWITCH HELP: Include symbolic links #
LIM.flcl.FLAM.DIR.ALIAS="" # TYPE: SWITCH HELP: Resolve data set aliases #
LIM.flcl.FLAM.DIR.HIDDEN="" # TYPE: SWITCH HELP: Include hidden files #
LIM.flcl.FLAM.DIR.RECURSIVE="" # TYPE: SWITCH HELP: Include sub directories #
LIM.flcl.FLAM.DIR.ARCHIVE="" # TYPE: SWITCH HELP: Include archives (FLAMFILEs, ZIP, ...) #
LIM.flcl.FLAM.DIR.ONEFILESYSTEM="" # TYPE: SWITCH HELP: Don't cross file system boundaries ↩
     #
LIM.flcl.FLAM.DIR.TAPE="" # TYPE: SWITCH HELP: Include files deposited on tapes #
LIM.flcl.FLAM.LOG="" # TYPE: OBJECT HELP: Logging Parameter #
LIM.flcl.FLAM.LOG.MESSAGE="" # TYPE: OBJECT HELP: Select type of messages to log #
LIM.flcl.FLAM.LOG.MESSAGE.NONE="" # TYPE: SWITCH HELP: No messages will be logged #
LIM.flcl.FLAM.LOG.MESSAGE.ERROR="" # TYPE: SWITCH HELP: Error messages will be logged #
LIM.flcl.FLAM.LOG.MESSAGE.ERRTRACE="" # TYPE: SWITCH HELP: Error trace will be logged #
LIM.flcl.FLAM.LOG.MESSAGE.WARNING="" # TYPE: SWITCH HELP: Warnings will be logged #
LIM.flcl.FLAM.LOG.MESSAGE.NOTICE="" # TYPE: SWITCH HELP: Notices will be logged #
LIM.flcl.FLAM.LOG.MESSAGE.DEBUG="" # TYPE: SWITCH HELP: Debug information will be logged #
LIM.flcl.FLAM.LOG.MESSAGE.ABOUT="" # TYPE: SWITCH HELP: About information will be logged #
LIM.flcl.FLAM.LOG.MESSAGE.VERSION="" # TYPE: SWITCH HELP: Version information will be  ↩
    logged #
LIM.flcl.FLAM.LOG.MESSAGE.INPUT="" # TYPE: SWITCH HELP: Input data will be logged ( ↩
    dangerous for passwords) #
LIM.flcl.FLAM.LOG.MESSAGE.PARAMETER="" # TYPE: SWITCH HELP: Parsed parameters will be  ↩
    logged #
LIM.flcl.FLAM.LOG.MESSAGE.STATISTIC="" # TYPE: SWITCH HELP: Statistics will be logged #
LIM.flcl.FLAM.LOG.MESSAGE.INFO="" # TYPE: SWITCH HELP: Requested information will be  ↩
    logged #
LIM.flcl.FLAM.LOG.MESSAGE.LICID="" # TYPE: SWITCH HELP: License ID will be logged #
LIM.flcl.FLAM.LOG.MESSAGE.LICENSE="" # TYPE: SWITCH HELP: License summary will be logged #
LIM.flcl.FLAM.LOG.MESSAGE.PROGRESS="" # TYPE: SWITCH HELP: Progress bar will be logged #
LIM.flcl.FLAM.LOG.MESSAGE.ALL="" # TYPE: SWITCH HELP: All messages are logged #
LIM.flcl.FLAM.LOG.MESSAGE.DEFAULT="" # TYPE: SWITCH HELP: Relevant messages are logged #
LIM.flcl.FLAM.LOG.MESSAGE.MINIMAL="" # TYPE: SWITCH HELP: Only important messages are  ↩
    logged #
LIM.flcl.FLAM.LOG.MESSAGE.ERRONLY="" # TYPE: SWITCH HELP: Only errors and error trace are  ↩
    logged #
LIM.flcl.FLAM.LOG.DESTINATION="" # TYPE: OVERLAY HELP: Destination for log messages #
LIM.flcl.FLAM.LOG.DESTINATION.STREAM="" # TYPE: OBJECT HELP: Log to a stream #
LIM.flcl.FLAM.LOG.DESTINATION.STREAM.FORMAT="" # TYPE: NUMBER HELP: Format of log message  ↩
    [STANDARD] #
```

```
LIM.flcl.FLAM.LOG.DESTINATION.STREAM.IDENT="" # TYPE: STRING HELP: Identifier used to identify log entries #
LIM.flcl.FLAM.LOG.DESTINATION.STREAM.STDOUT="" # TYPE: SWITCH HELP: Log messages to STDOUT instead of STDERR #
LIM.flcl.FLAM.LOG.DESTINATION.FILE="" # TYPE: OBJECT HELP: Log to a flat file #
LIM.flcl.FLAM.LOG.DESTINATION.FILE.FORMAT="" # TYPE: NUMBER HELP: Format of log message [STANDARD] #
LIM.flcl.FLAM.LOG.DESTINATION.FILE.IDENT="" # TYPE: STRING HELP: Identifier used to identify log entries #
LIM.flcl.FLAM.LOG.DESTINATION.FILE.NAME="" # TYPE: STRING HELP: Filename for appending log messages #
LIM.flcl.FLAM.LOG.DESTINATION.SYSTEM="" # TYPE: OBJECT HELP: Use system logging facility #
LIM.flcl.FLAM.LOG.DESTINATION.SYSTEM.IDENT="" # TYPE: STRING HELP: Identifier used to identify log entries #
LIM.flcl.XCHK.INPUT="" # TYPE: OBJECT HELP: Inbound parameter #
LIM.flcl.XCHK.INPUT.NET="" # TYPE: OVERLAY HELP: Network [NONE] #
LIM.flcl.XCHK.INPUT.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #
LIM.flcl.XCHK.INPUT.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
LIM.flcl.XCHK.INPUT.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
LIM.flcl.XCHK.INPUT.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
LIM.flcl.XCHK.INPUT.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #
LIM.flcl.XCHK.INPUT.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #
LIM.flcl.XCHK.INPUT.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
LIM.flcl.XCHK.INPUT.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
LIM.flcl.XCHK.INPUT.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
LIM.flcl.XCHK.INPUT.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #
LIM.flcl.XCHK.INPUT.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #
LIM.flcl.XCHK.INPUT.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
LIM.flcl.XCHK.INPUT.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
LIM.flcl.XCHK.INPUT.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
LIM.flcl.XCHK.INPUT.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #
LIM.flcl.XCHK.INPUT.SAV="" # TYPE: OVERLAY HELP: Storage [FILE] #
LIM.flcl.XCHK.INPUT.SAV.FILE="" # TYPE: OBJECT HELP: File handling #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO="" # TYPE: OVERLAY HELP: File IO [BLK] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.BLK="" # TYPE: OBJECT HELP: Read a file block-oriented ( mainly for open world) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.BLK.NAME="" # TYPE: STRING HELP: Name/URL of file to read [''==stdin] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.BLK.BLKSIZ="" # TYPE: NUMBER HELP: Block size for read operations [AUTO] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.BLK.DATTYP="" # TYPE: NUMBER HELP: Definition of data type for better handling [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.BLK.PRNCTR="" # TYPE: NUMBER HELP: Print control character handling if read ASA or MCC records [DETACH] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.BLK.FRCBLK="" # TYPE: SWITCH HELP: Enforce block orientation on record oriented devices [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.BLK.ADDDLM="" # TYPE: SWITCH HELP: Add a delimiter behind each record to produce a text block [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.BLK.SUBSYS="" # TYPE: OBJECT HELP: Specification for FIO subsystem (only for z/OS) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.BLK.SUBSYS.NAME="'FLAM'" # TYPE: STRING HELP: Name of the subsystem #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.BLK.SUBSYS.PARM="''" # TYPE: STRING HELP: Parameter list for the subsystem (comma separated) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.BLK.REMOVE="" # TYPE: SWITCH HELP: Remove original file after successful processing [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC="" # TYPE: OBJECT HELP: Read a file record-oriented ( mainly for mainframe world) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.NAME="" # TYPE: STRING HELP: Name/URL of file to read [''==stdin] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.RECMOD="" # TYPE: NUMBER HELP: Mode used to read records [STOP] #
```

```
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.PRNCTR="" # TYPE: NUMBER HELP: Print control  ↩
    character handling [DETACH] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.SUPPAD="" # TYPE: SWITCH HELP: Suppress trailing  ↩
    padding characters (useful for fixed records) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.PADCHR="" # TYPE: STRING HELP: Define padding  ↩
    character for suppression [automatic] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.FILORG="" # TYPE: NUMBER HELP: File organization [SEQ ↩
    ] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.BLKSIZ="" # TYPE: NUMBER HELP: Block size for read  ↩
    operations [AUTO] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.RECFMT="" # TYPE: NUMBER HELP: Record format for read ↩
     operations [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.RECLEN="" # TYPE: NUMBER HELP: Record length for read ↩
     operations [512] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.RECCNT="" # TYPE: NUMBER HELP: Amount of records  ↩
    handled in one read operation [128] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT="" # TYPE: OVERLAY HELP: Format of the length  ↩
    field for open platforms #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.INTEGER="" # TYPE: OBJECT HELP: Use integer  ↩
    format for the length field #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.INTEGER.ENDIAN="" # TYPE: NUMBER HELP: Byte  ↩
    order [SYSTEM] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.INTEGER.WIDTH="" # TYPE: NUMBER HELP: Bit  ↩
    width/rate [B32] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.INTEGER.INCLUDE="" # TYPE: SWITCH HELP:  ↩
    Include length field in length value [OFF] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.STRING="" # TYPE: OBJECT HELP: Use string  ↩
    format for the length field #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.STRING.CHRSET="" # TYPE: NUMBER HELP:  ↩
    Character set [SYSTEM] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.STRING.COUNT="" # TYPE: NUMBER HELP: Character ↩
     count [C06] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.STRING.BASE="" # TYPE: NUMBER HELP: String  ↩
    base [B10] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.STRING.INCLUDE="" # TYPE: SWITCH HELP: Include ↩
     length field in length value [OFF] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.BCD="" # TYPE: OBJECT HELP: Use BCD format for ↩
     the length field #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.BCD.WIDTH="" # TYPE: NUMBER HELP: Bit width/ ↩
    rate [B32] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.BCD.INCLUDE="" # TYPE: SWITCH HELP: Include  ↩
    length field in length value [OFF] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.HOST="" # TYPE: OBJECT HELP: Use host format ( ↩
    LLxx) for the length field #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.HOST.ENDIAN="" # TYPE: NUMBER HELP: Byte order ↩
     [BIG ENDIAN] for first 16 Bit #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.LENFMT.HOST.EXCLUDE="" # TYPE: SWITCH HELP: Record  ↩
    length field not part of the length value [OFF] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.CHRSET="" # TYPE: NUMBER HELP: Character set [auto  ↩
    detection] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.RECDLM="" # TYPE: STRING HELP: Delimiter used to  ↩
    define end of record #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.KEYPOS="" # TYPE: NUMBER HELP: Key position (>=1) for ↩
     index/key sequential data sets [0=NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.KEYLEN="" # TYPE: NUMBER HELP: Key length (>=1) for  ↩
    index/key sequential data sets [if KEYPOS>0 then 8] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.SUBSYS="" # TYPE: OBJECT HELP: Specification for FIO  ↩
    subsystem (only for z/OS) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.SUBSYS.NAME="'FLAM'" # TYPE: STRING HELP: Name of the ↩
     subsystem #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.SUBSYS.PARM="''" # TYPE: STRING HELP: Parameter list  ↩
    for the subsystem (comma separated) #
```

```
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.REC.REMOVE="" # TYPE: SWITCH HELP: Remove original file ↩
    after successful processing [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.TXT="" # TYPE: OBJECT HELP: Read a file text-oriented ( ↩
    mainly for open world) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.TXT.NAME="" # TYPE: STRING HELP: Name/URL of file to read ↩
     [''==stdin] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.TXT.RECMOD="" # TYPE: NUMBER HELP: Mode used to read ↩
    records [STOP] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.TXT.RECLEN="" # TYPE: NUMBER HELP: Record (Host) length / ↩
     Line (Unix/Win) length for read operations [512] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.TXT.RECCNT="" # TYPE: NUMBER HELP: Amount of records ↩
    handled in one read operation [128] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.TXT.PRNOUT="" # TYPE: NUMBER HELP: Amount of lines per ↩
    page if print control replacement active [0 = inactive] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.TXT.SUBSYS="" # TYPE: OBJECT HELP: Specification for FIO ↩
    subsystem (only for z/OS) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.TXT.SUBSYS.NAME="'FLAM'" # TYPE: STRING HELP: Name of the ↩
     subsystem #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.TXT.SUBSYS.PARM="''" # TYPE: STRING HELP: Parameter list ↩
    for the subsystem (comma separated) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.TXT.REMOVE="" # TYPE: SWITCH HELP: Remove original file ↩
    after successful processing [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4="" # TYPE: OBJECT HELP: Read a FLAMFILE record- ↩
    oriented #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.NAME="" # TYPE: STRING HELP: Name/URL of FLAMFILE to ↩
    read [required] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.MEMBER="" # TYPE: STRING HELP: Name or index of the ↩
    member in the FLAMFILE #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.RECLEN="" # TYPE: NUMBER HELP: Maximal record length ↩
    for read operations [512] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.RECCNT="" # TYPE: NUMBER HELP: Amount of records ↩
    handled in one read operation [128] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.TRUCAT="" # TYPE: SWITCH HELP: Allow truncation of ↩
    records [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.NONFIL="" # TYPE: SWITCH HELP: Switch to replace ↩
    original file name with FILEnnnn [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.PRNCTR="" # TYPE: NUMBER HELP: Print control ↩
    character handling [DETACH] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.DEVICE="" # TYPE: NUMBER HELP: Device used to store ↩
    the FLAMFILE [FLATFILE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.PASSWD="" # TYPE: STRING HELP: Passphrase to decrypt ↩
    the FLAMFILE [''] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.KMELIB="" # TYPE: STRING HELP: Library name for FKME ↩
    [''] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.KMEFUC="" # TYPE: STRING HELP: Function name of FKME ↩
    [''] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.KMEPAR="" # TYPE: STRING HELP: Parameter for the call ↩
     of FKME [''] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.EXTCMP="" # TYPE: STRING HELP: Name of user exit for ↩
    compression #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.EXTDCO="" # TYPE: STRING HELP: Name of user exit for ↩
    decompression #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.SECINF="" # TYPE: NUMBER HELP: Define handling of ↩
    secure information #
LIM.flcl.XCHK.INPUT.SAV.FILE.FIO.FL4.REMOVE="" # TYPE: SWITCH HELP: Remove FLAMFILE after ↩
    successful processing [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV="" # TYPE: OVERLAY HELP: Conversion [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BLK="" # TYPE: OBJECT HELP: Convert a record list to one ↩
    block (binary or text) of data #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BLK.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size ↩
    for preallocation [65536] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BLK.METHOD="" # TYPE: NUMBER HELP: Method for block/text ↩
```

```
    formatting [SYSTEM] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BLK.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM] ↩
     #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BLK.SUPTWS="" # TYPE: SWITCH HELP: Suppress trailing  ↩
    whitespace [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BLK.RPLTAB="" # TYPE: NUMBER HELP: Replace horizontal  ↩
    tabulators by spaces using this tab width [0 = no replacement] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BLK.RPLVTB="" # TYPE: NUMBER HELP: Replace vertical  ↩
    tabulators by new lines using this tab width [0 = no replacement] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BLK.RPLCTR="" # TYPE: NUMBER HELP: Replace remaining  ↩
    control characters [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC="" # TYPE: OBJECT HELP: Convert a block (binary or  ↩
    text) of data into a list of records #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC.METHOD="" # TYPE: NUMBER HELP: Method for record/text ↩
     formatting [DEFAULT] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM] ↩
     #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds,  ↩
    filling rest of page with blank lines assuming n lines per page [0 = no replacement] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC.RPLTAB="" # TYPE: NUMBER HELP: Replace horizontal  ↩
    tabulators by spaces using this tab width [0 = no replacement] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC.RPLVTB="" # TYPE: NUMBER HELP: Replace vertical  ↩
    tabulators by new lines using this tab width [0 = no replacement] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC.RPLCTR="" # TYPE: NUMBER HELP: Replace remaining  ↩
    control characters [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC.SUPTWS="" # TYPE: SWITCH HELP: Suppress trailing  ↩
    whitespaces [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC.NELDLM="" # TYPE: SWITCH HELP: Activate NEL (0x85) as ↩
     delimiter for ASCII character sets [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC.PADCHR="" # TYPE: NUMBER HELP: Padding character [0 ↩
    x00] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC.RECLEN="" # TYPE: NUMBER HELP: Length used to cut the ↩
     data block in records [512] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC.INICNT="" # TYPE: NUMBER HELP: Initial amount of  ↩
    records for preallocation [128] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.REC.SKPTXT="" # TYPE: SWITCH HELP: Convert only if record ↩
     length fields found in the data [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.GZP="" # TYPE: OBJECT HELP: Read ZLIB/GZIP/RCF1950-2  ↩
    compressed data #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.GZP.MEMBER="" # TYPE: STRING HELP: Index of the member in ↩
     the GZIP-File to read #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.GZP.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation[65536] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BZ2="" # TYPE: OBJECT HELP: Read BZIP2 compressed data #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BZ2.MEMBER="" # TYPE: STRING HELP: Index of the member in ↩
     the BZIP2-File to read #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BZ2.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BZ2.SMLSPC="" # TYPE: SWITCH HELP: Use small memory space ↩
     for decompression [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.LXZ="" # TYPE: OBJECT HELP: Read XZ (LZMA) compressed  ↩
    data #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.LXZ.MEMBER="" # TYPE: STRING HELP: Index of the member in ↩
     the XT-File to read #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.LXZ.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.ZIP="" # TYPE: OBJECT HELP: Read compressed data (GZIP,  ↩
    BZIP2, LZMA, ...) #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.ZIP.MEMBER="" # TYPE: STRING HELP: Index of the member in ↩
     the compressed file to read #
```

```
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.ZIP.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size for preallocation[65536] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR="" # TYPE: OBJECT HELP: Convert character sets (ASCII, EBCDIC, UNICODE) #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size for preallocation [65536] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.RECCNT="" # TYPE: NUMBER HELP: Initial amount of records for preallocation [128] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.METHOD="" # TYPE: NUMBER HELP: Enforces record or block-oriented conversion [AUTO] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.CASMOD="" # TYPE: NUMBER HELP: Define upper, lower or special case conversion [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.FROM="" # TYPE: STRING HELP: Conversion from this CCSID [auto detection] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.MODE="" # TYPE: NUMBER HELP: Define handling of non-convertible characters #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.TO="" # TYPE: STRING HELP: Conversion to this CCSID [UTF-8] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.WRTBOM="" # TYPE: SWITCH HELP: Write byte order mark in front of the output text #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.HDLBOM="" # TYPE: SWITCH HELP: Handle byte order change if get wrong BOM in the middle of data #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15) by line feed (0x25) on input #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.ELF2NL="" # TYPE: SWITCH HELP: Replace EBCDIC line feed (0x25) by new line (0x15) on output #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.SUBCHR="" # TYPE: NUMBER HELP: Default substitution character list [0x1A] (maximal 8 code points) #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.SYSTAB="" # TYPE: NUMBER HELP: Define a preloaded system substitution table #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.USRTAB="" # TYPE: STRING HELP: Name of the file containing the user conversion table #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.REPFIL="" # TYPE: STRING HELP: Name of the report file to log substitutions #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.SKPBIN="" # TYPE: SWITCH HELP: Skip character conversion if binary data is detected [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.CHR.SKPEQU="" # TYPE: SWITCH HELP: Skip character conversion if from-code equals to-code [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BAS="" # TYPE: OBJECT HELP: Convert Base16/32/64/85 encoded data to binary data #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BAS.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size for preallocation [65536] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BAS.RECCNT="" # TYPE: NUMBER HELP: Initial amount of records for preallocation [128] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BAS.METHOD="" # TYPE: NUMBER HELP: Base encoding selector [AUTO] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BAS.MODE="" # TYPE: NUMBER HELP: Enforces record or block-oriented conversion [BLOCK] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BAS.CHRSET="" # TYPE: NUMBER HELP: Character set [SYSTEM] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BAS.CRCCHK="" # TYPE: SWITCH HELP: Check CRC if available [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BAS.ARMCHK="" # TYPE: SWITCH HELP: Check ARMOR trailer if available [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BAS.RSTCHK="" # TYPE: SWITCH HELP: Check if a rest after encoded data [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.BAS.SKPNOB="" # TYPE: SWITCH HELP: Skip base decoding if no base encoding detected [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.HSH="" # TYPE: OBJECT HELP: Generation or verification of one way hash values #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.HSH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.HSH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum
```

```
     verification #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.HSH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex- ↩
    string #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.HSH.CHECK.FILE="" # TYPE: STRING HELP: File which  ↩
    contains Checksum as hex-string #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.HSH.FORMAT="" # TYPE: NUMBER HELP: Define printout format ↩
     [GNU] #
LIM.flcl.XCHK.INPUT.SAV.FILE.CNV.HSH.OUTPUT="" # TYPE: STRING HELP: Name of the output  ↩
    file [LOG] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT="" # TYPE: OVERLAY HELP: Formatting [BIN] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BIN="" # TYPE: OBJECT HELP: Format data in binary  ↩
    elements (block/records) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BIN.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BIN.RECCNT="" # TYPE: NUMBER HELP: Initial amount of  ↩
    records for preallocation [128] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BIN.TERASE="" # TYPE: SWITCH HELP: Erase data from memory ↩
     when no longer needed [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BIN.SEGINT="" # TYPE: NUMBER HELP: Segment integrity mode ↩
     [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BIN.LENMOD="" # TYPE: NUMBER HELP: Compression mode for  ↩
    length fields [COPY] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BIN.LENCNF="" # TYPE: NUMBER HELP: Encryption mode for  ↩
    length fields [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BIN.DATMOD="" # TYPE: NUMBER HELP: Compression mode for  ↩
    data [COPY] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BIN.DATCNF="" # TYPE: NUMBER HELP: Encryption mode for  ↩
    data [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BLK="" # TYPE: OBJECT HELP: Format data in one block  ↩
    element (all meta data lost) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BLK.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BLK.TERASE="" # TYPE: SWITCH HELP: Erase data from memory ↩
     when no longer needed [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BLK.SEGINT="" # TYPE: NUMBER HELP: Segment integrity mode ↩
     [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BLK.LENMOD="" # TYPE: NUMBER HELP: Compression mode for  ↩
    length fields [COPY] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BLK.LENCNF="" # TYPE: NUMBER HELP: Encryption mode for  ↩
    length fields [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BLK.DATMOD="" # TYPE: NUMBER HELP: Compression mode for  ↩
    data [COPY] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.BLK.DATCNF="" # TYPE: NUMBER HELP: Encryption mode for  ↩
    data [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.REC="" # TYPE: OBJECT HELP: Format data in several  ↩
    wrapped record elements (all meta data lost) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.REC.METHOD="" # TYPE: NUMBER HELP: Method for record  ↩
    formatting [AUTO] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.REC.WRPERR="" # TYPE: SWITCH HELP: If no method defined ( ↩
    auto detection) enforce an error if no length format (wrapping) detected [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.REC.RECLEN="" # TYPE: NUMBER HELP: Maximum length of a  ↩
    record [512] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.REC.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.REC.INICNT="" # TYPE: NUMBER HELP: Initial amount of  ↩
    records for preallocation [128] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.REC.TERASE="" # TYPE: SWITCH HELP: Erase data from memory ↩
     when no longer needed [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.REC.SEGINT="" # TYPE: NUMBER HELP: Segment integrity mode ↩
     [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.REC.LENMOD="" # TYPE: NUMBER HELP: Compression mode for  ↩
    length fields [COPY] #
```

```
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.REC.LENCNF="" # TYPE: NUMBER HELP: Encryption mode for  ↩
    length fields [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.REC.DATMOD="" # TYPE: NUMBER HELP: Compression mode for  ↩
    data [COPY] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.REC.DATCNF="" # TYPE: NUMBER HELP: Encryption mode for  ↩
    data [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT="" # TYPE: OBJECT HELP: Format data based on  ↩
    delimiters in text elements (record and rest) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.METHOD="" # TYPE: NUMBER HELP: Method for text  ↩
    formatting [DEFAULT] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.CHRSET="" # TYPE: NUMBER HELP: Character set [auto  ↩
    detection] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds,  ↩
    filling rest of page with blank lines assuming n lines per page [0 = no replacement] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.SUPTWS="" # TYPE: SWITCH HELP: Suppress trailing  ↩
    whitespaces [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.NELDLM="" # TYPE: SWITCH HELP: Activate NEL (0x85) as ↩
     delimiter for ASCII character sets [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.PADCHR="" # TYPE: NUMBER HELP: Padding character (is  ↩
    regarded as additional delimiter) [0x00] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.RECLEN="" # TYPE: NUMBER HELP: Maximum length of a  ↩
    text record (Host) / line (Unix/Win) [512] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.INICNT="" # TYPE: NUMBER HELP: Initial amount of  ↩
    records for preallocation [128] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.BINERR="" # TYPE: SWITCH HELP: Enforce an error if  ↩
    binary data detected [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.CHRERR="" # TYPE: SWITCH HELP: Enforce an error if  ↩
    character data without delimiter detected [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.TERASE="" # TYPE: SWITCH HELP: Erase data from memory ↩
     when no longer needed [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.SEGINT="" # TYPE: NUMBER HELP: Segment integrity mode ↩
     [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.LENMOD="" # TYPE: NUMBER HELP: Compression mode for  ↩
    length fields [COPY] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.LENCNF="" # TYPE: NUMBER HELP: Encryption mode for  ↩
    length fields [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.DATMOD="" # TYPE: NUMBER HELP: Compression mode for  ↩
    data [COPY] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.TXT.DATCNF="" # TYPE: NUMBER HELP: Encryption mode for  ↩
    data [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.XML="" # TYPE: OBJECT HELP: Format data stream in XML  ↩
    elements (tags, attributes, data, ...) #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.XML.CHRSET="" # TYPE: NUMBER HELP: Character set [auto] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.XML.FMTERR="" # TYPE: SWITCH HELP: Enforce a format error ↩
     if data contains only space or is empty [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.XML.NOCMNT="" # TYPE: SWITCH HELP: Ignore XML comments [ ↩
    FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.XML.BUFSIZ="" # TYPE: NUMBER HELP: Initial buffer size  ↩
    for preallocation [65536] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.XML.INICNT="" # TYPE: NUMBER HELP: Initial amount of  ↩
    records for preallocation [128] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.XML.TERASE="" # TYPE: SWITCH HELP: Erase data from memory ↩
     when no longer needed [FALSE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.XML.SEGINT="" # TYPE: NUMBER HELP: Segment integrity mode ↩
     [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.XML.LENMOD="" # TYPE: NUMBER HELP: Compression mode for  ↩
    length fields [COPY] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.XML.LENCNF="" # TYPE: NUMBER HELP: Encryption mode for  ↩
    length fields [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.XML.DATMOD="" # TYPE: NUMBER HELP: Compression mode for  ↩
```

```
    data [COPY] #
LIM.flcl.XCHK.INPUT.SAV.FILE.FMT.XML.DATCNF="" # TYPE: NUMBER HELP: Encryption mode for ↩
    data [NONE] #
LIM.flcl.XCHK.INPUT.SAV.FILE.ENV="" # TYPE: OBJECT HELP: Environment #
LIM.flcl.XCHK.INPUT.SAV.FILE.ENV.KEYWORD="" # TYPE: STRING HELP: Keyword of environment ↩
    variable #
LIM.flcl.XCHK.INPUT.SAV.FILE.ENV.VALUE="" # TYPE: STRING HELP: Value of environment ↩
    variable #
LIM.flcl.XCHK.DIR="" # TYPE: OBJECT HELP: Defines how to handle directories and various ↩
    file types if wildcards used #
LIM.flcl.XCHK.DIR.LINK="" # TYPE: SWITCH HELP: Include symbolic links #
LIM.flcl.XCHK.DIR.ALIAS="" # TYPE: SWITCH HELP: Resolve data set aliases #
LIM.flcl.XCHK.DIR.HIDDEN="" # TYPE: SWITCH HELP: Include hidden files #
LIM.flcl.XCHK.DIR.RECURSIVE="" # TYPE: SWITCH HELP: Include sub directories #
LIM.flcl.XCHK.DIR.ARCHIVE="" # TYPE: SWITCH HELP: Include archives (FLAMFILEs, ZIP, ...) #
LIM.flcl.XCHK.DIR.ONEFILESYSTEM="" # TYPE: SWITCH HELP: Don't cross file system boundaries ↩
     #
LIM.flcl.XCHK.DIR.TAPE="" # TYPE: SWITCH HELP: Include files deposited on tapes #
LIM.flcl.XCHK.LOG="" # TYPE: OBJECT HELP: Logging Parameter #
LIM.flcl.XCHK.LOG.MESSAGE="" # TYPE: OBJECT HELP: Select type of messages to log #
LIM.flcl.XCHK.LOG.MESSAGE.NONE="" # TYPE: SWITCH HELP: No messages will be logged #
LIM.flcl.XCHK.LOG.MESSAGE.ERROR="" # TYPE: SWITCH HELP: Error messages will be logged #
LIM.flcl.XCHK.LOG.MESSAGE.ERRTRACE="" # TYPE: SWITCH HELP: Error trace will be logged #
LIM.flcl.XCHK.LOG.MESSAGE.WARNING="" # TYPE: SWITCH HELP: Warnings will be logged #
LIM.flcl.XCHK.LOG.MESSAGE.NOTICE="" # TYPE: SWITCH HELP: Notices will be logged #
LIM.flcl.XCHK.LOG.MESSAGE.DEBUG="" # TYPE: SWITCH HELP: Debug information will be logged #
LIM.flcl.XCHK.LOG.MESSAGE.ABOUT="" # TYPE: SWITCH HELP: About information will be logged #
LIM.flcl.XCHK.LOG.MESSAGE.VERSION="" # TYPE: SWITCH HELP: Version information will be ↩
    logged #
LIM.flcl.XCHK.LOG.MESSAGE.INPUT="" # TYPE: SWITCH HELP: Input data will be logged ( ↩
    dangerous for passwords) #
LIM.flcl.XCHK.LOG.MESSAGE.PARAMETER="" # TYPE: SWITCH HELP: Parsed parameters will be ↩
    logged #
LIM.flcl.XCHK.LOG.MESSAGE.STATISTIC="" # TYPE: SWITCH HELP: Statistics will be logged #
LIM.flcl.XCHK.LOG.MESSAGE.INFO="" # TYPE: SWITCH HELP: Requested information will be ↩
    logged #
LIM.flcl.XCHK.LOG.MESSAGE.LICID="" # TYPE: SWITCH HELP: License ID will be logged #
LIM.flcl.XCHK.LOG.MESSAGE.LICENSE="" # TYPE: SWITCH HELP: License summary will be logged #
LIM.flcl.XCHK.LOG.MESSAGE.PROGRESS="" # TYPE: SWITCH HELP: Progress bar will be logged #
LIM.flcl.XCHK.LOG.MESSAGE.ALL="" # TYPE: SWITCH HELP: All messages are logged #
LIM.flcl.XCHK.LOG.MESSAGE.DEFAULT="" # TYPE: SWITCH HELP: Relevant messages are logged #
LIM.flcl.XCHK.LOG.MESSAGE.MINIMAL="" # TYPE: SWITCH HELP: Only important messages are ↩
    logged #
LIM.flcl.XCHK.LOG.MESSAGE.ERRONLY="" # TYPE: SWITCH HELP: Only errors and error trace are ↩
    logged #
LIM.flcl.XCHK.LOG.DESTINATION="" # TYPE: OVERLAY HELP: Destination for log messages #
LIM.flcl.XCHK.LOG.DESTINATION.STREAM="" # TYPE: OBJECT HELP: Log to a stream #
LIM.flcl.XCHK.LOG.DESTINATION.STREAM.FORMAT="" # TYPE: NUMBER HELP: Format of log message ↩
    [STANDARD] #
LIM.flcl.XCHK.LOG.DESTINATION.STREAM.IDENT="" # TYPE: STRING HELP: Identifier used to ↩
    identify log entries #
LIM.flcl.XCHK.LOG.DESTINATION.STREAM.STDOUT="" # TYPE: SWITCH HELP: Log messages to STDOUT ↩
     instead of STDERR #
LIM.flcl.XCHK.LOG.DESTINATION.FILE="" # TYPE: OBJECT HELP: Log to a flat file #
LIM.flcl.XCHK.LOG.DESTINATION.FILE.FORMAT="" # TYPE: NUMBER HELP: Format of log message [ ↩
    STANDARD] #
LIM.flcl.XCHK.LOG.DESTINATION.FILE.IDENT="" # TYPE: STRING HELP: Identifier used to ↩
    identify log entries #
LIM.flcl.XCHK.LOG.DESTINATION.FILE.NAME="" # TYPE: STRING HELP: Filename for appending log ↩
     messages #
LIM.flcl.XCHK.LOG.DESTINATION.SYSTEM="" # TYPE: OBJECT HELP: Use system logging facility #
LIM.flcl.XCHK.LOG.DESTINATION.SYSTEM.IDENT="" # TYPE: STRING HELP: Identifier used to ↩
```

```
    identify log entries #
LIM.flcl.HASH.FILE="" # TYPE: STRING HELP: Name/URL of file to hash [''==origin.ext] #
LIM.flcl.HASH.DIR="" # TYPE: OBJECT HELP: Defines how to handle directories and various ↩
    file types if wildcards used #
LIM.flcl.HASH.DIR.LINK="" # TYPE: SWITCH HELP: Include symbolic links #
LIM.flcl.HASH.DIR.ALIAS="" # TYPE: SWITCH HELP: Resolve data set aliases #
LIM.flcl.HASH.DIR.HIDDEN="" # TYPE: SWITCH HELP: Include hidden files #
LIM.flcl.HASH.DIR.RECURSIVE="" # TYPE: SWITCH HELP: Include sub directories #
LIM.flcl.HASH.DIR.ARCHIVE="" # TYPE: SWITCH HELP: Include archives (FLAMFILEs, ZIP, ...) #
LIM.flcl.HASH.DIR.ONEFILESYSTEM="" # TYPE: SWITCH HELP: Don't cross file system boundaries ↩
     #
LIM.flcl.HASH.DIR.TAPE="" # TYPE: SWITCH HELP: Include files deposited on tapes #
LIM.flcl.HASH.LOG="" # TYPE: OBJECT HELP: Logging Parameter #
LIM.flcl.HASH.LOG.MESSAGE="" # TYPE: OBJECT HELP: Select type of messages to log #
LIM.flcl.HASH.LOG.MESSAGE.NONE="" # TYPE: SWITCH HELP: No messages will be logged #
LIM.flcl.HASH.LOG.MESSAGE.ERROR="" # TYPE: SWITCH HELP: Error messages will be logged #
LIM.flcl.HASH.LOG.MESSAGE.ERRTRACE="" # TYPE: SWITCH HELP: Error trace will be logged #
LIM.flcl.HASH.LOG.MESSAGE.WARNING="" # TYPE: SWITCH HELP: Warnings will be logged #
LIM.flcl.HASH.LOG.MESSAGE.NOTICE="" # TYPE: SWITCH HELP: Notices will be logged #
LIM.flcl.HASH.LOG.MESSAGE.DEBUG="" # TYPE: SWITCH HELP: Debug information will be logged #
LIM.flcl.HASH.LOG.MESSAGE.ABOUT="" # TYPE: SWITCH HELP: About information will be logged #
LIM.flcl.HASH.LOG.MESSAGE.VERSION="" # TYPE: SWITCH HELP: Version information will be ↩
    logged #
LIM.flcl.HASH.LOG.MESSAGE.INPUT="" # TYPE: SWITCH HELP: Input data will be logged ( ↩
    dangerous for passwords) #
LIM.flcl.HASH.LOG.MESSAGE.PARAMETER="" # TYPE: SWITCH HELP: Parsed parameters will be ↩
    logged #
LIM.flcl.HASH.LOG.MESSAGE.STATISTIC="" # TYPE: SWITCH HELP: Statistics will be logged #
LIM.flcl.HASH.LOG.MESSAGE.INFO="" # TYPE: SWITCH HELP: Requested information will be ↩
    logged #
LIM.flcl.HASH.LOG.MESSAGE.LICID="" # TYPE: SWITCH HELP: License ID will be logged #
LIM.flcl.HASH.LOG.MESSAGE.LICENSE="" # TYPE: SWITCH HELP: License summary will be logged #
LIM.flcl.HASH.LOG.MESSAGE.PROGRESS="" # TYPE: SWITCH HELP: Progress bar will be logged #
LIM.flcl.HASH.LOG.MESSAGE.ALL="" # TYPE: SWITCH HELP: All messages are logged #
LIM.flcl.HASH.LOG.MESSAGE.DEFAULT="" # TYPE: SWITCH HELP: Relevant messages are logged #
LIM.flcl.HASH.LOG.MESSAGE.MINIMAL="" # TYPE: SWITCH HELP: Only important messages are ↩
    logged #
LIM.flcl.HASH.LOG.MESSAGE.ERRONLY="" # TYPE: SWITCH HELP: Only errors and error trace are ↩
    logged #
LIM.flcl.HASH.LOG.DESTINATION="" # TYPE: OVERLAY HELP: Destination for log messages #
LIM.flcl.HASH.LOG.DESTINATION.STREAM="" # TYPE: OBJECT HELP: Log to a stream #
LIM.flcl.HASH.LOG.DESTINATION.STREAM.FORMAT="" # TYPE: NUMBER HELP: Format of log message ↩
    [STANDARD] #
LIM.flcl.HASH.LOG.DESTINATION.STREAM.IDENT="" # TYPE: STRING HELP: Identifier used to ↩
    identify log entries #
LIM.flcl.HASH.LOG.DESTINATION.STREAM.STDOUT="" # TYPE: SWITCH HELP: Log messages to STDOUT ↩
     instead of STDERR #
LIM.flcl.HASH.LOG.DESTINATION.FILE="" # TYPE: OBJECT HELP: Log to a flat file #
LIM.flcl.HASH.LOG.DESTINATION.FILE.FORMAT="" # TYPE: NUMBER HELP: Format of log message [ ↩
    STANDARD] #
LIM.flcl.HASH.LOG.DESTINATION.FILE.IDENT="" # TYPE: STRING HELP: Identifier used to ↩
    identify log entries #
LIM.flcl.HASH.LOG.DESTINATION.FILE.NAME="" # TYPE: STRING HELP: Filename for appending log ↩
     messages #
LIM.flcl.HASH.LOG.DESTINATION.SYSTEM="" # TYPE: OBJECT HELP: Use system logging facility #
LIM.flcl.HASH.LOG.DESTINATION.SYSTEM.IDENT="" # TYPE: STRING HELP: Identifier used to ↩
    identify log entries #
LIM.flcl.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
LIM.flcl.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum verification #
LIM.flcl.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string #
LIM.flcl.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains Checksum as hex- ↩
    string #
```

```
 LIM.flcl.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] #
 LIM.flcl.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG] #
 LIM.flcl.DIFF.READ="" # TYPE: OVERLAY HELP: Parameter to read data #
 LIM.flcl.DIFF.READ.BINARY="" # TYPE: OBJECT HELP: Read binary data from a file #

# DESCRIPTION for LIM.flcl.DIFF.READ.BINARY.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.READ.BINARY.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read  ↩
    access #

# DESCRIPTION for LIM.flcl.DIFF.READ.BINARY.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.BINARY.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.DIFF.READ.BINARY.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.BINARY.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.READ.BINARY.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.BINARY.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
    optional] #

# DESCRIPTION for LIM.flcl.DIFF.READ.BINARY.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.
```

```
For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.BINARY.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.DIFF.READ.BINARY.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.BINARY.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.READ.BINARY.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.BINARY.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.READ.BINARY.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.BINARY.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.READ.BINARY.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] ↩
      #
 LIM.flcl.DIFF.READ.BINARY.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.DIFF.READ.BINARY.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.BINARY.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.DIFF.READ.BINARY.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''==stdin ↩
     ] #
 LIM.flcl.DIFF.READ.BINARY.BLKSIZE="" # TYPE: NUMBER HELP: Block size to read [AUTO] #
 LIM.flcl.DIFF.READ.BINARY.DECODE="" # TYPE: NUMBER HELP: Decode encoded, encrypted or  ↩
     compressed data [0=NONE] #
 LIM.flcl.DIFF.READ.BINARY.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
 LIM.flcl.DIFF.READ.BINARY.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password  ↩
     decryption #
 LIM.flcl.DIFF.READ.BINARY.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to  ↩
     encrypt the FLAMFILE ['********'] #
 LIM.flcl.DIFF.READ.BINARY.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management  ↩
     extension (FKME) #
 LIM.flcl.DIFF.READ.BINARY.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for  ↩
     FKME ['libfkme'] #
 LIM.flcl.DIFF.READ.BINARY.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of  ↩
     FKME ['FKMESTD0'] #
 LIM.flcl.DIFF.READ.BINARY.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for  ↩
     the call of FKME [''] #
 LIM.flcl.DIFF.READ.BINARY.FDECODE="" # TYPE: OVERLAY HELP: Additional enforced decoding [ ↩
     NONE] #
 LIM.flcl.DIFF.READ.BINARY.FDECODE.BASE64="" # TYPE: SWITCH HELP: Base64 decoding (RFC4648) ↩
      #
 LIM.flcl.DIFF.READ.BINARY.FDECODE.BASE32="" # TYPE: SWITCH HELP: Base32 decoding (RFC4648) ↩
      #
 LIM.flcl.DIFF.READ.BINARY.FDECODE.BASE16="" # TYPE: SWITCH HELP: Base16 decoding (RFC4648) ↩
      #
```

```
 LIM.flcl.DIFF.READ.BINARY.RLDECO="" # TYPE: SWITCH HELP: Activate an additional record ↩
     length decoding [FALSE] #
 LIM.flcl.DIFF.READ.BINARY.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character ↩
     handling [DETACH] #
 LIM.flcl.DIFF.READ.BINARY.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO ↩
     subsystem (only for z/OS) #
 LIM.flcl.DIFF.READ.BINARY.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the ↩
     subsystem #
 LIM.flcl.DIFF.READ.BINARY.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list for the ↩
     subsystem (comma separated) #
 LIM.flcl.DIFF.READ.BINARY.FRCBLK="" # TYPE: SWITCH HELP: Enforce block orientation on ↩
     record oriented devices [FALSE] #
 LIM.flcl.DIFF.READ.BINARY.REMOVE="" # TYPE: SWITCH HELP: Remove original file after ↩
     successful processing [FALSE] #
 LIM.flcl.DIFF.READ.BINARY.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable ↩
     FL_PLATFORM for read operation #
 LIM.flcl.DIFF.READ.BINARY.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash ↩
      values of input data #
 LIM.flcl.DIFF.READ.BINARY.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.DIFF.READ.BINARY.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum ↩
     verification #
 LIM.flcl.DIFF.READ.BINARY.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string ↩
     #
 LIM.flcl.DIFF.READ.BINARY.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains ↩
     Checksum as hex-string #
 LIM.flcl.DIFF.READ.BINARY.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] ↩
      #
 LIM.flcl.DIFF.READ.BINARY.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG ↩
     ] #
 LIM.flcl.DIFF.READ.BINARY.CHECK="" # TYPE: SWITCH HELP: Activate additional optional ↩
     verifications [FALSE] #
 LIM.flcl.DIFF.READ.CHARACTER="" # TYPE: OBJECT HELP: Read character data from a file #

# DESCRIPTION for LIM.flcl.DIFF.READ.CHARACTER.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.READ.CHARACTER.NET="" # TYPE: OVERLAY HELP: Network parameter for remote ↩
     read access #

# DESCRIPTION for LIM.flcl.DIFF.READ.CHARACTER.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
```

```
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.CHARACTER.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.DIFF.READ.CHARACTER.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.CHARACTER.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.READ.CHARACTER.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.CHARACTER.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.READ.CHARACTER.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.CHARACTER.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.DIFF.READ.CHARACTER.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.CHARACTER.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.READ.CHARACTER.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.CHARACTER.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.READ.CHARACTER.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.CHARACTER.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.READ.CHARACTER.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name  ↩
     [""] #
 LIM.flcl.DIFF.READ.CHARACTER.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.DIFF.READ.CHARACTER.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.CHARACTER.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
```

```
    optional] #

LIM.flcl.DIFF.READ.CHARACTER.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''== ↩
    stdin] #
LIM.flcl.DIFF.READ.CHARACTER.BLKSIZE="" # TYPE: NUMBER HELP: Block size for read  ↩
    operations [AUTO] #
LIM.flcl.DIFF.READ.CHARACTER.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID [ ↩
    auto detection] #
LIM.flcl.DIFF.READ.CHARACTER.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get  ↩
    wrong BOM in the middle of data #
LIM.flcl.DIFF.READ.CHARACTER.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15) ↩
     by line feed (0x25) on input #
LIM.flcl.DIFF.READ.CHARACTER.DECODE="ALWAYS" # TYPE: NUMBER HELP: Decode encoded data  ↩
    first (remove base encodings) #
LIM.flcl.DIFF.READ.CHARACTER.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE]  ↩
    #
LIM.flcl.DIFF.READ.CHARACTER.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password  ↩
    decryption #
LIM.flcl.DIFF.READ.CHARACTER.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to  ↩
    encrypt the FLAMFILE ['********'] #
LIM.flcl.DIFF.READ.CHARACTER.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management ↩
     extension (FKME) #
LIM.flcl.DIFF.READ.CHARACTER.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name  ↩
    for FKME ['libfkme'] #
LIM.flcl.DIFF.READ.CHARACTER.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name  ↩
    of FKME ['FKMESTD0'] #
LIM.flcl.DIFF.READ.CHARACTER.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for ↩
     the call of FKME [''] #
LIM.flcl.DIFF.READ.CHARACTER.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character  ↩
    handling [DETACH] #
LIM.flcl.DIFF.READ.CHARACTER.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO  ↩
    subsystem (only for z/OS) #
LIM.flcl.DIFF.READ.CHARACTER.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the  ↩
    subsystem #
LIM.flcl.DIFF.READ.CHARACTER.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list for  ↩
    the subsystem (comma separated) #
LIM.flcl.DIFF.READ.CHARACTER.FRCBLK="" # TYPE: SWITCH HELP: Enforce block orientation on  ↩
    record oriented devices [FALSE] #
LIM.flcl.DIFF.READ.CHARACTER.REMOVE="" # TYPE: SWITCH HELP: Remove original file after  ↩
    successful processing [FALSE] #
LIM.flcl.DIFF.READ.CHARACTER.LANG="" # TYPE: STRING HELP: Redefine environment variable  ↩
    LANG for read operation #
LIM.flcl.DIFF.READ.CHARACTER.PLATFORM="" # TYPE: STRING HELP: Redefine environment  ↩
    variable FL_PLATFORM for read operation #
LIM.flcl.DIFF.READ.CHARACTER.HASH="" # TYPE: OBJECT HELP: Generation or check of one way  ↩
    hash values of input data #
LIM.flcl.DIFF.READ.CHARACTER.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
LIM.flcl.DIFF.READ.CHARACTER.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
LIM.flcl.DIFF.READ.CHARACTER.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex- ↩
    string #
LIM.flcl.DIFF.READ.CHARACTER.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
LIM.flcl.DIFF.READ.CHARACTER.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [ ↩
    GNU] #
LIM.flcl.DIFF.READ.CHARACTER.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [ ↩
    LOG] #
LIM.flcl.DIFF.READ.CHARACTER.CHECK="" # TYPE: SWITCH HELP: Activate additional optional  ↩
    verifications [FALSE] #
LIM.flcl.DIFF.READ.TEXT="" # TYPE: OBJECT HELP: Read text data from a file #

# DESCRIPTION for LIM.flcl.DIFF.READ.TEXT.NET:
```

```
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.READ.TEXT.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read  ↩
     access #

# DESCRIPTION for LIM.flcl.DIFF.READ.TEXT.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.TEXT.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.DIFF.READ.TEXT.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.TEXT.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.READ.TEXT.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.TEXT.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

# DESCRIPTION for LIM.flcl.DIFF.READ.TEXT.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.TEXT.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #
```

```
 LIM.flcl.DIFF.READ.TEXT.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.TEXT.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.READ.TEXT.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.TEXT.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

# DESCRIPTION for LIM.flcl.DIFF.READ.TEXT.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.TEXT.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.READ.TEXT.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
 LIM.flcl.DIFF.READ.TEXT.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.DIFF.READ.TEXT.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.TEXT.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

 LIM.flcl.DIFF.READ.TEXT.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''==stdin]  ↩
     #
 LIM.flcl.DIFF.READ.TEXT.BLKSIZE="" # TYPE: NUMBER HELP: Block size for read operations [ ↩
     AUTO] #
 LIM.flcl.DIFF.READ.TEXT.RECLENGTH="" # TYPE: NUMBER HELP: Maximum record length (Host) /  ↩
     line length (Unix/Win) for text parsing [512] #
 LIM.flcl.DIFF.READ.TEXT.SUPTWS="" # TYPE: SWITCH HELP: Suppress trailing whitespaces [ ↩
     FALSE] #
 LIM.flcl.DIFF.READ.TEXT.NELDLM="" # TYPE: SWITCH HELP: Activate NEL (0x85) as delimiter  ↩
     for ASCII character sets [FALSE] #
 LIM.flcl.DIFF.READ.TEXT.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID [auto  ↩
     detection] #
 LIM.flcl.DIFF.READ.TEXT.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get wrong  ↩
     BOM in the middle of data #
 LIM.flcl.DIFF.READ.TEXT.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15) by  ↩
     line feed (0x25) on input #
 LIM.flcl.DIFF.READ.TEXT.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds, filling rest of ↩
      page with blank lines assuming n lines per page [0 = no replacement] #
 LIM.flcl.DIFF.READ.TEXT.PADCHAR="" # TYPE: NUMBER HELP: Padding character to be regarded  ↩
     as additional delimiter [0x00] #
 LIM.flcl.DIFF.READ.TEXT.DECODE="ALWAYS" # TYPE: NUMBER HELP: Decode encoded data first ( ↩
     remove base encodings) #
 LIM.flcl.DIFF.READ.TEXT.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
 LIM.flcl.DIFF.READ.TEXT.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password decryption ↩
      #
 LIM.flcl.DIFF.READ.TEXT.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to  ↩
     encrypt the FLAMFILE ['********'] #
 LIM.flcl.DIFF.READ.TEXT.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management  ↩
     extension (FKME) #
 LIM.flcl.DIFF.READ.TEXT.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for  ↩
     FKME ['libfkme'] #
 LIM.flcl.DIFF.READ.TEXT.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of  ↩
     FKME ['FKMESTD0'] #
```

```
 LIM.flcl.DIFF.READ.TEXT.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for the  ↩
     call of FKME [’’] #
 LIM.flcl.DIFF.READ.TEXT.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character  ↩
     handling [DETACH] #
 LIM.flcl.DIFF.READ.TEXT.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO subsystem  ↩
     (only for z/OS) #
 LIM.flcl.DIFF.READ.TEXT.SUBSYSTEM.NAME="’FLAM’" # TYPE: STRING HELP: Name of the subsystem ↩
      #
 LIM.flcl.DIFF.READ.TEXT.SUBSYSTEM.PARM="’’" # TYPE: STRING HELP: Parameter list for the  ↩
     subsystem (comma separated) #
 LIM.flcl.DIFF.READ.TEXT.FRCBLK="" # TYPE: SWITCH HELP: Enforce block orientation on record ↩
      oriented devices [FALSE] #
 LIM.flcl.DIFF.READ.TEXT.REMOVE="" # TYPE: SWITCH HELP: Remove original file after  ↩
     successful processing [FALSE] #
 LIM.flcl.DIFF.READ.TEXT.BINERROR="" # TYPE: SWITCH HELP: Enforce an error if binary data  ↩
     detected [FALSE] #
 LIM.flcl.DIFF.READ.TEXT.CHRERROR="" # TYPE: SWITCH HELP: Enforce an error if character  ↩
     data without delimiter detected [FALSE] #
 LIM.flcl.DIFF.READ.TEXT.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG  ↩
     for read operation #
 LIM.flcl.DIFF.READ.TEXT.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
     FL_PLATFORM for read operation #
 LIM.flcl.DIFF.READ.TEXT.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash  ↩
     values of input data #
 LIM.flcl.DIFF.READ.TEXT.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.DIFF.READ.TEXT.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
     verification #
 LIM.flcl.DIFF.READ.TEXT.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string #
 LIM.flcl.DIFF.READ.TEXT.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
     Checksum as hex-string #
 LIM.flcl.DIFF.READ.TEXT.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] #
 LIM.flcl.DIFF.READ.TEXT.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG]  ↩
     #
 LIM.flcl.DIFF.READ.TEXT.CHECK="" # TYPE: SWITCH HELP: Activate additional optional  ↩
     verifications [FALSE] #
 LIM.flcl.DIFF.READ.XML="" # TYPE: OBJECT HELP: Read XML data from a file #

# DESCRIPTION for LIM.flcl.DIFF.READ.XML.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.READ.XML.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read  ↩
     access #

# DESCRIPTION for LIM.flcl.DIFF.READ.XML.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.
```

```
The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.XML.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.DIFF.READ.XML.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.XML.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.READ.XML.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.XML.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional] ↩
      #

# DESCRIPTION for LIM.flcl.DIFF.READ.XML.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.XML.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.DIFF.READ.XML.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.XML.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.READ.XML.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.XML.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] ↩
      #

# DESCRIPTION for LIM.flcl.DIFF.READ.XML.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.XML.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.READ.XML.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
```

```
LIM.flcl.DIFF.READ.XML.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
LIM.flcl.DIFF.READ.XML.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
LIM.flcl.DIFF.READ.XML.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] ↩
     #

LIM.flcl.DIFF.READ.XML.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''==stdin] #
LIM.flcl.DIFF.READ.XML.BLKSIZE="" # TYPE: NUMBER HELP: Block size for read operations [ ↩
    AUTO] #
LIM.flcl.DIFF.READ.XML.NOCMNT="" # TYPE: SWITCH HELP: Ignore XML comments [FALSE] #
LIM.flcl.DIFF.READ.XML.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID [no  ↩
    character conversion, UTF-8/US-ASCII expected] #
LIM.flcl.DIFF.READ.XML.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get wrong  ↩
    BOM in the middle of data #
LIM.flcl.DIFF.READ.XML.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15) by  ↩
    line feed (0x25) on input #
LIM.flcl.DIFF.READ.XML.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds, filling rest of  ↩
    page with blank lines assuming n lines per page [0 = no replacement] #
LIM.flcl.DIFF.READ.XML.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
LIM.flcl.DIFF.READ.XML.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password decryption  ↩
    #
LIM.flcl.DIFF.READ.XML.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to  ↩
    encrypt the FLAMFILE ['********'] #
LIM.flcl.DIFF.READ.XML.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management  ↩
    extension (FKME) #
LIM.flcl.DIFF.READ.XML.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for FKME ↩
     ['libfkme'] #
LIM.flcl.DIFF.READ.XML.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of  ↩
    FKME ['FKMESTD0'] #
LIM.flcl.DIFF.READ.XML.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for the  ↩
    call of FKME [''] #
LIM.flcl.DIFF.READ.XML.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO subsystem ( ↩
    only for z/OS) #
LIM.flcl.DIFF.READ.XML.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the subsystem  ↩
    #
LIM.flcl.DIFF.READ.XML.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list for the  ↩
    subsystem (comma separated) #
LIM.flcl.DIFF.READ.XML.FRCBLK="" # TYPE: SWITCH HELP: Enforce block orientation on record  ↩
    oriented devices [FALSE] #
LIM.flcl.DIFF.READ.XML.REMOVE="" # TYPE: SWITCH HELP: Remove original file after  ↩
    successful processing [FALSE] #
LIM.flcl.DIFF.READ.XML.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG for ↩
     read operation #
LIM.flcl.DIFF.READ.XML.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
    FL_PLATFORM for read operation #
LIM.flcl.DIFF.READ.XML.HASH="" # TYPE: OBJECT HELP: Generation or check  of one way hash  ↩
    values of input data #
LIM.flcl.DIFF.READ.XML.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
LIM.flcl.DIFF.READ.XML.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
LIM.flcl.DIFF.READ.XML.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string #
LIM.flcl.DIFF.READ.XML.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
LIM.flcl.DIFF.READ.XML.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] #
LIM.flcl.DIFF.READ.XML.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG] #
LIM.flcl.DIFF.READ.XML.CHECK="" # TYPE: SWITCH HELP: Activate additional optional  ↩
    verifications [FALSE] #
LIM.flcl.DIFF.READ.RECORD="" # TYPE: OBJECT HELP: Read records from a file #

# DESCRIPTION for LIM.flcl.DIFF.READ.RECORD.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.
```

```
This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.READ.RECORD.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read  ↩
     access #

# DESCRIPTION for LIM.flcl.DIFF.READ.RECORD.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.RECORD.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.DIFF.READ.RECORD.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.RECORD.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.READ.RECORD.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.RECORD.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.READ.RECORD.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.RECORD.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.DIFF.READ.RECORD.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.RECORD.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
```

```
 LIM.flcl.DIFF.READ.RECORD.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.RECORD.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
    optional] #

# DESCRIPTION for LIM.flcl.DIFF.READ.RECORD.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.RECORD.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.READ.RECORD.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] ↩
     #
 LIM.flcl.DIFF.READ.RECORD.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.DIFF.READ.RECORD.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.RECORD.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
    optional] #

 LIM.flcl.DIFF.READ.RECORD.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''==stdin ↩
    ] #
 LIM.flcl.DIFF.READ.RECORD.RECMODE="" # TYPE: NUMBER HELP: Mode used to read records [STOP] ↩
     #
 LIM.flcl.DIFF.READ.RECORD.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character  ↩
    handling [DETACH] #
 LIM.flcl.DIFF.READ.RECORD.SUPPADDING="" # TYPE: SWITCH HELP: Suppress trailing padding  ↩
    characters (useful for fix records) #
 LIM.flcl.DIFF.READ.RECORD.ORGANIZATION="" # TYPE: NUMBER HELP: File organization [catalog/ ↩
    SEQ] #
 LIM.flcl.DIFF.READ.RECORD.BLKSIZE="" # TYPE: NUMBER HELP: Block size for read operations [ ↩
    catalog/AUTO] #
 LIM.flcl.DIFF.READ.RECORD.RECFORMAT="" # TYPE: NUMBER HELP: Record format for read  ↩
    operation [catalog/VAR] #
 LIM.flcl.DIFF.READ.RECORD.RECLENGTH="" # TYPE: NUMBER HELP: Maximal record length for read ↩
     operation [catalog/512] #
 LIM.flcl.DIFF.READ.RECORD.RECDELIM="" # TYPE: STRING HELP: Delimiter used to define end of ↩
     record (if RECF=DLM used) #
 LIM.flcl.DIFF.READ.RECORD.RECCOUNT="" # TYPE: NUMBER HELP: Amount of records handled in  ↩
    one read operation [catalog/128] #
 LIM.flcl.DIFF.READ.RECORD.LENFORMAT="" # TYPE: OVERLAY HELP: Format of the length field  ↩
    for open platforms [word] #
 LIM.flcl.DIFF.READ.RECORD.LENFORMAT.INTEGER="" # TYPE: OBJECT HELP: Use integer format for ↩
     the length field #
 LIM.flcl.DIFF.READ.RECORD.LENFORMAT.INTEGER.ENDIAN="" # TYPE: NUMBER HELP: Byte order [ ↩
    SYSTEM] #
 LIM.flcl.DIFF.READ.RECORD.LENFORMAT.INTEGER.WIDTH="" # TYPE: NUMBER HELP: Bit width/rate [ ↩
    B32] #
 LIM.flcl.DIFF.READ.RECORD.LENFORMAT.INTEGER.INCLUDE="" # TYPE: SWITCH HELP: Include length ↩
     field in length value [OFF] #
 LIM.flcl.DIFF.READ.RECORD.LENFORMAT.STRING="" # TYPE: OBJECT HELP: Use string format for  ↩
    the length field #
 LIM.flcl.DIFF.READ.RECORD.LENFORMAT.STRING.CHRSET="" # TYPE: NUMBER HELP: Character set [ ↩
    SYSTEM] #
 LIM.flcl.DIFF.READ.RECORD.LENFORMAT.STRING.COUNT="" # TYPE: NUMBER HELP: Character count [ ↩
```

```
    C06] #
LIM.flcl.DIFF.READ.RECORD.LENFORMAT.STRING.BASE="" # TYPE: NUMBER HELP: String base [B10]  ↩
    #
LIM.flcl.DIFF.READ.RECORD.LENFORMAT.STRING.INCLUDE="" # TYPE: SWITCH HELP: Include length  ↩
    field in length value [OFF] #
LIM.flcl.DIFF.READ.RECORD.LENFORMAT.BCD="" # TYPE: OBJECT HELP: Use BCD format for the  ↩
    length field #
LIM.flcl.DIFF.READ.RECORD.LENFORMAT.BCD.WIDTH="" # TYPE: NUMBER HELP: Bit width/rate [B32] ↩
     #
LIM.flcl.DIFF.READ.RECORD.LENFORMAT.BCD.INCLUDE="" # TYPE: SWITCH HELP: Include length  ↩
    field in length value [OFF] #
LIM.flcl.DIFF.READ.RECORD.LENFORMAT.HOST="" # TYPE: OBJECT HELP: Use host format (LLxx)  ↩
    for the length field #
LIM.flcl.DIFF.READ.RECORD.LENFORMAT.HOST.ENDIAN="" # TYPE: NUMBER HELP: Byte order [BIG  ↩
    ENDIAN] for first 16 Bit #
LIM.flcl.DIFF.READ.RECORD.LENFORMAT.HOST.EXCLUDE="" # TYPE: SWITCH HELP: Record length  ↩
    field not part of the length value [OFF] #
LIM.flcl.DIFF.READ.RECORD.KEYPOSITION="" # TYPE: NUMBER HELP: Key position (>=1) for index ↩
    /key sequential data sets [0=NONE] #
LIM.flcl.DIFF.READ.RECORD.KEYLENGTH="" # TYPE: NUMBER HELP: Key length (>=1) for index/key ↩
     sequential data sets [if KEYPOS>0 then 8] #
LIM.flcl.DIFF.READ.RECORD.DECODE="" # TYPE: NUMBER HELP: Decode encoded, encrypted or  ↩
    compressed data [0=NONE] #
LIM.flcl.DIFF.READ.RECORD.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
LIM.flcl.DIFF.READ.RECORD.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password  ↩
    decryption #
LIM.flcl.DIFF.READ.RECORD.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to  ↩
    encrypt the FLAMFILE ['********'] #
LIM.flcl.DIFF.READ.RECORD.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management  ↩
    extension (FKME) #
LIM.flcl.DIFF.READ.RECORD.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for  ↩
    FKME ['libfkme'] #
LIM.flcl.DIFF.READ.RECORD.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of  ↩
    FKME ['FKMESTD0'] #
LIM.flcl.DIFF.READ.RECORD.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for  ↩
    the call of FKME [''] #
LIM.flcl.DIFF.READ.RECORD.FDECODE="" # TYPE: OVERLAY HELP: Additional enforced decoding [ ↩
    NONE] #
LIM.flcl.DIFF.READ.RECORD.FDECODE.BASE64="" # TYPE: SWITCH HELP: Base64 decoding (RFC4648) ↩
     #
LIM.flcl.DIFF.READ.RECORD.FDECODE.BASE32="" # TYPE: SWITCH HELP: Base32 decoding (RFC4648) ↩
     #
LIM.flcl.DIFF.READ.RECORD.FDECODE.BASE16="" # TYPE: SWITCH HELP: Base16 decoding (RFC4648) ↩
     #
LIM.flcl.DIFF.READ.RECORD.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID ( ↩
    enables text handling if set) #
LIM.flcl.DIFF.READ.RECORD.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get  ↩
    wrong BOM in the middle of data #
LIM.flcl.DIFF.READ.RECORD.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO  ↩
    subsystem (only for z/OS) #
LIM.flcl.DIFF.READ.RECORD.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the  ↩
    subsystem #
LIM.flcl.DIFF.READ.RECORD.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list for the  ↩
    subsystem (comma separated) #
LIM.flcl.DIFF.READ.RECORD.REMOVE="" # TYPE: SWITCH HELP: Remove original file after  ↩
    successful processing [FALSE] #
LIM.flcl.DIFF.READ.RECORD.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG  ↩
    for read operation #
LIM.flcl.DIFF.READ.RECORD.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
    FL_PLATFORM for read operation #
LIM.flcl.DIFF.READ.RECORD.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash ↩
     values of input data #
```

```
 LIM.flcl.DIFF.READ.RECORD.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.DIFF.READ.RECORD.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
     verification #
 LIM.flcl.DIFF.READ.RECORD.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string  ↩
     #
 LIM.flcl.DIFF.READ.RECORD.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
     Checksum as hex-string #
 LIM.flcl.DIFF.READ.RECORD.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] ↩
      #
 LIM.flcl.DIFF.READ.RECORD.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG ↩
     ] #
 LIM.flcl.DIFF.READ.RECORD.CHECK="" # TYPE: SWITCH HELP: Activate additional optional  ↩
     verifications [FALSE] #
 LIM.flcl.DIFF.READ.FLAM4="" # TYPE: OBJECT HELP: Read records from a FLAM4FILE #

# DESCRIPTION for LIM.flcl.DIFF.READ.FLAM4.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.READ.FLAM4.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read  ↩
     access #

# DESCRIPTION for LIM.flcl.DIFF.READ.FLAM4.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.FLAM4.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.DIFF.READ.FLAM4.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.FLAM4.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.READ.FLAM4.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.FLAM4.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.READ.FLAM4.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
```

```
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.FLAM4.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.DIFF.READ.FLAM4.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.FLAM4.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.READ.FLAM4.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.FLAM4.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.READ.FLAM4.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.FLAM4.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.READ.FLAM4.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""]  ↩
     #
 LIM.flcl.DIFF.READ.FLAM4.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.DIFF.READ.FLAM4.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.FLAM4.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.DIFF.READ.FLAM4.FILE="" # TYPE: STRING HELP: Name/URL of the FLAMFILE to read [ ↩
     required] #
 LIM.flcl.DIFF.READ.FLAM4.MEMBER="" # TYPE: STRING HELP: Name or Index of the member in the ↩
      FLAMFILE to read #
 LIM.flcl.DIFF.READ.FLAM4.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character  ↩
     handling [DETACH] #
 LIM.flcl.DIFF.READ.FLAM4.RECCOUNT="" # TYPE: NUMBER HELP: Amount of records handled in one ↩
      read operation [128] #
 LIM.flcl.DIFF.READ.FLAM4.RECLENGTH="" # TYPE: NUMBER HELP: Maximal record length for read  ↩
     operation [512] #
 LIM.flcl.DIFF.READ.FLAM4.TRUNCATE="" # TYPE: SWITCH HELP: Allow truncation of records [ ↩
     FALSE] #
 LIM.flcl.DIFF.READ.FLAM4.PASSWORD="" # TYPE: STRING HELP: Passphrase to decrypt the  ↩
     FLAMFILE [''] #
 LIM.flcl.DIFF.READ.FLAM4.KMLIBRARY="" # TYPE: STRING HELP: Library name for FKME [''] #
 LIM.flcl.DIFF.READ.FLAM4.KMFUNCTION="" # TYPE: STRING HELP: Function name of FKME [''] #
```

```
 LIM.flcl.DIFF.READ.FLAM4.KMPARAMETER="" # TYPE: STRING HELP: Parameter for the call of  ↩
    FKME [''] #
 LIM.flcl.DIFF.READ.FLAM4.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID (enables ↩
     text handling if set) #
 LIM.flcl.DIFF.READ.FLAM4.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get wrong ↩
     BOM in the middle of data #
 LIM.flcl.DIFF.READ.FLAM4.REMOVE="" # TYPE: SWITCH HELP: Remove FLAMFILE after successful  ↩
    processing [FALSE] #
 LIM.flcl.DIFF.READ.FLAM4.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG  ↩
    for read operation #
 LIM.flcl.DIFF.READ.FLAM4.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
    FL_PLATFORM for read operation #
 LIM.flcl.DIFF.READ.FLAM4.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash  ↩
    values of input data #
 LIM.flcl.DIFF.READ.FLAM4.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.DIFF.READ.FLAM4.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
 LIM.flcl.DIFF.READ.FLAM4.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string #
 LIM.flcl.DIFF.READ.FLAM4.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
 LIM.flcl.DIFF.READ.FLAM4.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU]  ↩
    #
 LIM.flcl.DIFF.READ.FLAM4.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG] ↩
     #
 LIM.flcl.DIFF.READ.AUTO="" # TYPE: OBJECT HELP: Read binary, text or XML data record- or  ↩
    block-oriented #

# DESCRIPTION for LIM.flcl.DIFF.READ.AUTO.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.READ.AUTO.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read  ↩
    access #

# DESCRIPTION for LIM.flcl.DIFF.READ.AUTO.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.AUTO.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #
```

```
 LIM.flcl.DIFF.READ.AUTO.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.AUTO.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.READ.AUTO.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.AUTO.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

# DESCRIPTION for LIM.flcl.DIFF.READ.AUTO.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.AUTO.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.DIFF.READ.AUTO.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.READ.AUTO.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.READ.AUTO.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.AUTO.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

# DESCRIPTION for LIM.flcl.DIFF.READ.AUTO.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.READ.AUTO.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.READ.AUTO.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
 LIM.flcl.DIFF.READ.AUTO.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.DIFF.READ.AUTO.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.READ.AUTO.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional ↩
     ] #

 LIM.flcl.DIFF.READ.AUTO.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''==stdin]  ↩
     #
 LIM.flcl.DIFF.READ.AUTO.BLKSIZE="" # TYPE: NUMBER HELP: Block size to read [AUTO] #
 LIM.flcl.DIFF.READ.AUTO.RECLENGTH="" # TYPE: NUMBER HELP: Maximum record length (Host) /  ↩
     line length (Unix/Win) for text parsing [512] #
```

```
 LIM.flcl.DIFF.READ.AUTO.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds, filling rest of ↩
      page with blank lines assuming n lines per page [0 = no replacement] #
 LIM.flcl.DIFF.READ.AUTO.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID [auto  ↩
     detection] #
 LIM.flcl.DIFF.READ.AUTO.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get wrong  ↩
     BOM in the middle of data #
 LIM.flcl.DIFF.READ.AUTO.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15) by  ↩
     line feed (0x25) on input #
 LIM.flcl.DIFF.READ.AUTO.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
 LIM.flcl.DIFF.READ.AUTO.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password decryption ↩
      #
 LIM.flcl.DIFF.READ.AUTO.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to  ↩
     encrypt the FLAMFILE ['********'] #
 LIM.flcl.DIFF.READ.AUTO.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management  ↩
     extension (FKME) #
 LIM.flcl.DIFF.READ.AUTO.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for  ↩
     FKME ['libfkme'] #
 LIM.flcl.DIFF.READ.AUTO.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of  ↩
     FKME ['FKMESTD0'] #
 LIM.flcl.DIFF.READ.AUTO.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for the  ↩
     call of FKME [''] #
 LIM.flcl.DIFF.READ.AUTO.REMOVE="" # TYPE: SWITCH HELP: Remove original file after  ↩
     successful processing [FALSE] #
 LIM.flcl.DIFF.READ.AUTO.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG  ↩
     for read operation #
 LIM.flcl.DIFF.READ.AUTO.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
     FL_PLATFORM for read operation #
 LIM.flcl.DIFF.READ.AUTO.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash  ↩
     values of input data #
 LIM.flcl.DIFF.READ.AUTO.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.DIFF.READ.AUTO.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
     verification #
 LIM.flcl.DIFF.READ.AUTO.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string #
 LIM.flcl.DIFF.READ.AUTO.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
     Checksum as hex-string #
 LIM.flcl.DIFF.READ.AUTO.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] #
 LIM.flcl.DIFF.READ.AUTO.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG]  ↩
     #
 LIM.flcl.DIFF.READ.AUTO.CHECK="" # TYPE: SWITCH HELP: Activate additional optional  ↩
     verifications [FALSE] #
 LIM.flcl.DIFF.READ.FILE="" # TYPE: STRING HELP: Abbreviation for read.auto(file=...) #
 LIM.flcl.DIFF.COMPARE="" # TYPE: OVERLAY HELP: Parameter to compare with data read #
 LIM.flcl.DIFF.COMPARE.BINARY="" # TYPE: OBJECT HELP: Read binary data from a file #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.BINARY.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.COMPARE.BINARY.NET="" # TYPE: OVERLAY HELP: Network parameter for remote  ↩
     read access #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.BINARY.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
```

```
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.BINARY.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.DIFF.COMPARE.BINARY.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.COMPARE.BINARY.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.BINARY.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.BINARY.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.BINARY.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.BINARY.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.DIFF.COMPARE.BINARY.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.COMPARE.BINARY.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.BINARY.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.BINARY.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.BINARY.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
```

```
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.BINARY.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.COMPARE.BINARY.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name ↩
     [""] #
 LIM.flcl.DIFF.COMPARE.BINARY.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.DIFF.COMPARE.BINARY.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.BINARY.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.DIFF.COMPARE.BINARY.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''== ↩
     stdin] #
 LIM.flcl.DIFF.COMPARE.BINARY.BLKSIZE="" # TYPE: NUMBER HELP: Block size to read [AUTO] #
 LIM.flcl.DIFF.COMPARE.BINARY.DECODE="" # TYPE: NUMBER HELP: Decode encoded, encrypted or ↩
     compressed data [0=NONE] #
 LIM.flcl.DIFF.COMPARE.BINARY.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] ↩
     #
 LIM.flcl.DIFF.COMPARE.BINARY.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password ↩
     decryption #
 LIM.flcl.DIFF.COMPARE.BINARY.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to ↩
     encrypt the FLAMFILE ['********'] #
 LIM.flcl.DIFF.COMPARE.BINARY.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management ↩
      extension (FKME) #
 LIM.flcl.DIFF.COMPARE.BINARY.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name ↩
     for FKME ['libfkme'] #
 LIM.flcl.DIFF.COMPARE.BINARY.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name ↩
     of FKME ['FKMESTD0'] #
 LIM.flcl.DIFF.COMPARE.BINARY.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for ↩
      the call of FKME [''] #
 LIM.flcl.DIFF.COMPARE.BINARY.FDECODE="" # TYPE: OVERLAY HELP: Additional enforced decoding ↩
      [NONE] #
 LIM.flcl.DIFF.COMPARE.BINARY.FDECODE.BASE64="" # TYPE: SWITCH HELP: Base64 decoding ( ↩
     RFC4648) #
 LIM.flcl.DIFF.COMPARE.BINARY.FDECODE.BASE32="" # TYPE: SWITCH HELP: Base32 decoding ( ↩
     RFC4648) #
 LIM.flcl.DIFF.COMPARE.BINARY.FDECODE.BASE16="" # TYPE: SWITCH HELP: Base16 decoding ( ↩
     RFC4648) #
 LIM.flcl.DIFF.COMPARE.BINARY.RLDECO="" # TYPE: SWITCH HELP: Activate an additional record ↩
     length decoding [FALSE] #
 LIM.flcl.DIFF.COMPARE.BINARY.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character ↩
     handling [DETACH] #
 LIM.flcl.DIFF.COMPARE.BINARY.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO ↩
     subsystem (only for z/OS) #
 LIM.flcl.DIFF.COMPARE.BINARY.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the ↩
     subsystem #
 LIM.flcl.DIFF.COMPARE.BINARY.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list for ↩
     the subsystem (comma separated) #
 LIM.flcl.DIFF.COMPARE.BINARY.FRCBLK="" # TYPE: SWITCH HELP: Enforce block orientation on ↩
     record oriented devices [FALSE] #
 LIM.flcl.DIFF.COMPARE.BINARY.REMOVE="" # TYPE: SWITCH HELP: Remove original file after ↩
     successful processing [FALSE] #
 LIM.flcl.DIFF.COMPARE.BINARY.PLATFORM="" # TYPE: STRING HELP: Redefine environment ↩
     variable FL_PLATFORM for read operation #
 LIM.flcl.DIFF.COMPARE.BINARY.HASH="" # TYPE: OBJECT HELP: Generation or check of one way ↩
     hash values of input data #
 LIM.flcl.DIFF.COMPARE.BINARY.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.DIFF.COMPARE.BINARY.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum ↩
     verification #
 LIM.flcl.DIFF.COMPARE.BINARY.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex- ↩
```

```
    string #
 LIM.flcl.DIFF.COMPARE.BINARY.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
 LIM.flcl.DIFF.COMPARE.BINARY.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [ ↩
    GNU] #
 LIM.flcl.DIFF.COMPARE.BINARY.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [ ↩
    LOG] #
 LIM.flcl.DIFF.COMPARE.BINARY.CHECK="" # TYPE: SWITCH HELP: Activate additional optional  ↩
    verifications [FALSE] #
 LIM.flcl.DIFF.COMPARE.CHARACTER="" # TYPE: OBJECT HELP: Read character data from a file #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.CHARACTER.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.COMPARE.CHARACTER.NET="" # TYPE: OVERLAY HELP: Network parameter for remote  ↩
    read access #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.CHARACTER.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost ↩
    ] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional]  ↩
    #
 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
    optional] #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.CHARACTER.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.
```

```
Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost ↩
     ] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional]  ↩
     #
 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.CHARACTER.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager  ↩
     name [""] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional]  ↩
     #
 LIM.flcl.DIFF.COMPARE.CHARACTER.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.DIFF.COMPARE.CHARACTER.FILE="" # TYPE: STRING HELP: Name/URL of file to read  ↩
     [''==stdin] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.BLKSIZE="" # TYPE: NUMBER HELP: Block size for read  ↩
     operations [AUTO] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID [ ↩
     auto detection] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.BOM="" # TYPE: SWITCH HELP: Handle byte order change if  ↩
     get wrong BOM in the middle of data #
 LIM.flcl.DIFF.COMPARE.CHARACTER.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0 ↩
     x15) by line feed (0x25) on input #
 LIM.flcl.DIFF.COMPARE.CHARACTER.DECODE="ALWAYS" # TYPE: NUMBER HELP: Decode encoded data  ↩
     first (remove base encodings) #
 LIM.flcl.DIFF.COMPARE.CHARACTER.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [ ↩
     NONE] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password  ↩
```

```
    decryption #
 LIM.flcl.DIFF.COMPARE.CHARACTER.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase ↩
    to encrypt the FLAMFILE ['********'] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key ↩
    management extension (FKME) #
 LIM.flcl.DIFF.COMPARE.CHARACTER.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name ↩
     for FKME ['libfkme'] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function ↩
    name of FKME ['FKMESTD0'] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter ↩
    for the call of FKME [''] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character ↩
     handling [DETACH] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO ↩
    subsystem (only for z/OS) #
 LIM.flcl.DIFF.COMPARE.CHARACTER.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the ↩
    subsystem #
 LIM.flcl.DIFF.COMPARE.CHARACTER.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list ↩
    for the subsystem (comma separated) #
 LIM.flcl.DIFF.COMPARE.CHARACTER.FRCBLK="" # TYPE: SWITCH HELP: Enforce block orientation ↩
    on record oriented devices [FALSE] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.REMOVE="" # TYPE: SWITCH HELP: Remove original file after ↩
    successful processing [FALSE] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.LANG="" # TYPE: STRING HELP: Redefine environment variable ↩
     LANG for read operation #
 LIM.flcl.DIFF.COMPARE.CHARACTER.PLATFORM="" # TYPE: STRING HELP: Redefine environment ↩
    variable FL_PLATFORM for read operation #
 LIM.flcl.DIFF.COMPARE.CHARACTER.HASH="" # TYPE: OBJECT HELP: Generation or check of one ↩
    way hash values of input data #
 LIM.flcl.DIFF.COMPARE.CHARACTER.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum ↩
     verification #
 LIM.flcl.DIFF.COMPARE.CHARACTER.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex- ↩
    string #
 LIM.flcl.DIFF.COMPARE.CHARACTER.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which ↩
    contains Checksum as hex-string #
 LIM.flcl.DIFF.COMPARE.CHARACTER.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format ↩
     [GNU] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output ↩
    file [LOG] #
 LIM.flcl.DIFF.COMPARE.CHARACTER.CHECK="" # TYPE: SWITCH HELP: Activate additional optional ↩
     verifications [FALSE] #
 LIM.flcl.DIFF.COMPARE.TEXT="" # TYPE: OBJECT HELP: Read text data from a file #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.TEXT.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.COMPARE.TEXT.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read ↩
    access #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.TEXT.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
```

```
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.TEXT.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.DIFF.COMPARE.TEXT.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.COMPARE.TEXT.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.TEXT.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.TEXT.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.TEXT.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.TEXT.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.DIFF.COMPARE.TEXT.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.COMPARE.TEXT.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.TEXT.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.TEXT.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.TEXT.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
```

```
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.TEXT.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.COMPARE.TEXT.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name  ↩
     [""] #
 LIM.flcl.DIFF.COMPARE.TEXT.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.DIFF.COMPARE.TEXT.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.TEXT.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.DIFF.COMPARE.TEXT.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''== ↩
     stdin] #
 LIM.flcl.DIFF.COMPARE.TEXT.BLKSIZE="" # TYPE: NUMBER HELP: Block size for read operations  ↩
     [AUTO] #
 LIM.flcl.DIFF.COMPARE.TEXT.RECLENGTH="" # TYPE: NUMBER HELP: Maximum record length (Host)  ↩
     / line length (Unix/Win) for text parsing [512] #
 LIM.flcl.DIFF.COMPARE.TEXT.SUPTWS="" # TYPE: SWITCH HELP: Suppress trailing whitespaces [ ↩
     FALSE] #
 LIM.flcl.DIFF.COMPARE.TEXT.NELDLM="" # TYPE: SWITCH HELP: Activate NEL (0x85) as delimiter ↩
      for ASCII character sets [FALSE] #
 LIM.flcl.DIFF.COMPARE.TEXT.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID [auto  ↩
     detection] #
 LIM.flcl.DIFF.COMPARE.TEXT.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get  ↩
     wrong BOM in the middle of data #
 LIM.flcl.DIFF.COMPARE.TEXT.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15)  ↩
     by line feed (0x25) on input #
 LIM.flcl.DIFF.COMPARE.TEXT.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds, filling rest ↩
      of page with blank lines assuming n lines per page [0 = no replacement] #
 LIM.flcl.DIFF.COMPARE.TEXT.PADCHAR="" # TYPE: NUMBER HELP: Padding character to be  ↩
     regarded as additional delimiter [0x00] #
 LIM.flcl.DIFF.COMPARE.TEXT.DECODE="ALWAYS" # TYPE: NUMBER HELP: Decode encoded data first  ↩
     (remove base encodings) #
 LIM.flcl.DIFF.COMPARE.TEXT.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
 LIM.flcl.DIFF.COMPARE.TEXT.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password  ↩
     decryption #
 LIM.flcl.DIFF.COMPARE.TEXT.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to  ↩
     encrypt the FLAMFILE ['********'] #
 LIM.flcl.DIFF.COMPARE.TEXT.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management  ↩
     extension (FKME) #
 LIM.flcl.DIFF.COMPARE.TEXT.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for  ↩
     FKME ['libfkme'] #
 LIM.flcl.DIFF.COMPARE.TEXT.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of ↩
      FKME ['FKMESTD0'] #
 LIM.flcl.DIFF.COMPARE.TEXT.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for  ↩
     the call of FKME [''] #
 LIM.flcl.DIFF.COMPARE.TEXT.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character  ↩
     handling [DETACH] #
 LIM.flcl.DIFF.COMPARE.TEXT.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO  ↩
     subsystem (only for z/OS) #
 LIM.flcl.DIFF.COMPARE.TEXT.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the  ↩
     subsystem #
 LIM.flcl.DIFF.COMPARE.TEXT.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list for the ↩
      subsystem (comma separated) #
 LIM.flcl.DIFF.COMPARE.TEXT.FRCBLK="" # TYPE: SWITCH HELP: Enforce block orientation on  ↩
     record oriented devices [FALSE] #
 LIM.flcl.DIFF.COMPARE.TEXT.REMOVE="" # TYPE: SWITCH HELP: Remove original file after  ↩
     successful processing [FALSE] #
 LIM.flcl.DIFF.COMPARE.TEXT.BINERROR="" # TYPE: SWITCH HELP: Enforce an error if binary  ↩
     data detected [FALSE] #
```

```
 LIM.flcl.DIFF.COMPARE.TEXT.CHRERROR="" # TYPE: SWITCH HELP: Enforce an error if character ↩
     data without delimiter detected [FALSE] #
 LIM.flcl.DIFF.COMPARE.TEXT.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG ↩
      for read operation #
 LIM.flcl.DIFF.COMPARE.TEXT.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable ↩
     FL_PLATFORM for read operation #
 LIM.flcl.DIFF.COMPARE.TEXT.HASH="" # TYPE: OBJECT HELP: Generation or check of one way ↩
     hash values of input data #
 LIM.flcl.DIFF.COMPARE.TEXT.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.DIFF.COMPARE.TEXT.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum ↩
     verification #
 LIM.flcl.DIFF.COMPARE.TEXT.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string ↩
      #
 LIM.flcl.DIFF.COMPARE.TEXT.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains ↩
     Checksum as hex-string #
 LIM.flcl.DIFF.COMPARE.TEXT.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU ↩
     ] #
 LIM.flcl.DIFF.COMPARE.TEXT.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [ ↩
     LOG] #
 LIM.flcl.DIFF.COMPARE.TEXT.CHECK="" # TYPE: SWITCH HELP: Activate additional optional ↩
     verifications [FALSE] #
 LIM.flcl.DIFF.COMPARE.XML="" # TYPE: OBJECT HELP: Read XML data from a file #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.XML.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.COMPARE.XML.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read ↩
     access #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.XML.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.XML.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.DIFF.COMPARE.XML.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.COMPARE.XML.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.XML.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.XML.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
```

```
    optional] #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.XML.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.XML.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.DIFF.COMPARE.XML.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.COMPARE.XML.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.XML.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.XML.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
    optional] #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.XML.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.XML.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.COMPARE.XML.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] ↩
     #
 LIM.flcl.DIFF.COMPARE.XML.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.DIFF.COMPARE.XML.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.XML.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
    optional] #

 LIM.flcl.DIFF.COMPARE.XML.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''==stdin ↩
    ] #
 LIM.flcl.DIFF.COMPARE.XML.BLKSIZE="" # TYPE: NUMBER HELP: Block size for read operations [ ↩
    AUTO] #
 LIM.flcl.DIFF.COMPARE.XML.NOCMNT="" # TYPE: SWITCH HELP: Ignore XML comments [FALSE] #
 LIM.flcl.DIFF.COMPARE.XML.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID [no  ↩
    character conversion, UTF-8/US-ASCII expected] #
 LIM.flcl.DIFF.COMPARE.XML.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get  ↩
```

```
    wrong BOM in the middle of data #
 LIM.flcl.DIFF.COMPARE.XML.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15) by ↩
     line feed (0x25) on input #
 LIM.flcl.DIFF.COMPARE.XML.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds, filling rest  ↩
    of page with blank lines assuming n lines per page [0 = no replacement] #
 LIM.flcl.DIFF.COMPARE.XML.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
 LIM.flcl.DIFF.COMPARE.XML.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password  ↩
    decryption #
 LIM.flcl.DIFF.COMPARE.XML.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to  ↩
    encrypt the FLAMFILE ['********'] #
 LIM.flcl.DIFF.COMPARE.XML.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management  ↩
    extension (FKME) #
 LIM.flcl.DIFF.COMPARE.XML.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for  ↩
    FKME ['libfkme'] #
 LIM.flcl.DIFF.COMPARE.XML.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of  ↩
    FKME ['FKMESTD0'] #
 LIM.flcl.DIFF.COMPARE.XML.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for  ↩
    the call of FKME [''] #
 LIM.flcl.DIFF.COMPARE.XML.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO  ↩
    subsystem (only for z/OS) #
 LIM.flcl.DIFF.COMPARE.XML.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the  ↩
    subsystem #
 LIM.flcl.DIFF.COMPARE.XML.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list for the  ↩
    subsystem (comma separated) #
 LIM.flcl.DIFF.COMPARE.XML.FRCBLK="" # TYPE: SWITCH HELP: Enforce block orientation on  ↩
    record oriented devices [FALSE] #
 LIM.flcl.DIFF.COMPARE.XML.REMOVE="" # TYPE: SWITCH HELP: Remove original file after  ↩
    successful processing [FALSE] #
 LIM.flcl.DIFF.COMPARE.XML.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG  ↩
    for read operation #
 LIM.flcl.DIFF.COMPARE.XML.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
    FL_PLATFORM for read operation #
 LIM.flcl.DIFF.COMPARE.XML.HASH="" # TYPE: OBJECT HELP: Generation or check  of one way  ↩
    hash values of input data #
 LIM.flcl.DIFF.COMPARE.XML.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.DIFF.COMPARE.XML.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
 LIM.flcl.DIFF.COMPARE.XML.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string  ↩
    #
 LIM.flcl.DIFF.COMPARE.XML.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
 LIM.flcl.DIFF.COMPARE.XML.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU] ↩
     #
 LIM.flcl.DIFF.COMPARE.XML.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [LOG ↩
    ] #
 LIM.flcl.DIFF.COMPARE.XML.CHECK="" # TYPE: SWITCH HELP: Activate additional optional  ↩
    verifications [FALSE] #
 LIM.flcl.DIFF.COMPARE.RECORD="" # TYPE: OBJECT HELP: Read records from a file #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.RECORD.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.COMPARE.RECORD.NET="" # TYPE: OVERLAY HELP: Network parameter for remote  ↩
    read access #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.RECORD.NET.IPN:
```

```
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.RECORD.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.DIFF.COMPARE.RECORD.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.COMPARE.RECORD.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.RECORD.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.RECORD.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.RECORD.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.RECORD.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.DIFF.COMPARE.RECORD.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.COMPARE.RECORD.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.RECORD.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.RECORD.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.RECORD.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
```

```
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.RECORD.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.COMPARE.RECORD.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name  ↩
     [""] #
 LIM.flcl.DIFF.COMPARE.RECORD.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.DIFF.COMPARE.RECORD.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.RECORD.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.DIFF.COMPARE.RECORD.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''== ↩
     stdin] #
 LIM.flcl.DIFF.COMPARE.RECORD.RECMODE="" # TYPE: NUMBER HELP: Mode used to read records [ ↩
     STOP] #
 LIM.flcl.DIFF.COMPARE.RECORD.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character  ↩
     handling [DETACH] #
 LIM.flcl.DIFF.COMPARE.RECORD.SUPPADDING="" # TYPE: SWITCH HELP: Suppress trailing padding  ↩
     characters (useful for fix records) #
 LIM.flcl.DIFF.COMPARE.RECORD.ORGANIZATION="" # TYPE: NUMBER HELP: File organization [ ↩
     catalog/SEQ] #
 LIM.flcl.DIFF.COMPARE.RECORD.BLKSIZE="" # TYPE: NUMBER HELP: Block size for read  ↩
     operations [catalog/AUTO] #
 LIM.flcl.DIFF.COMPARE.RECORD.RECFORMAT="" # TYPE: NUMBER HELP: Record format for read  ↩
     operation [catalog/VAR] #
 LIM.flcl.DIFF.COMPARE.RECORD.RECLENGTH="" # TYPE: NUMBER HELP: Maximal record length for  ↩
     read operation [catalog/512] #
 LIM.flcl.DIFF.COMPARE.RECORD.RECDELIM="" # TYPE: STRING HELP: Delimiter used to define end ↩
      of record (if RECF=DLM used) #
 LIM.flcl.DIFF.COMPARE.RECORD.RECCOUNT="" # TYPE: NUMBER HELP: Amount of records handled in ↩
      one read operation [catalog/128] #
 LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT="" # TYPE: OVERLAY HELP: Format of the length field ↩
      for open platforms [word] #
 LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.INTEGER="" # TYPE: OBJECT HELP: Use integer format  ↩
     for the length field #
 LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.INTEGER.ENDIAN="" # TYPE: NUMBER HELP: Byte order [ ↩
     SYSTEM] #
 LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.INTEGER.WIDTH="" # TYPE: NUMBER HELP: Bit width/ ↩
     rate [B32] #
 LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.INTEGER.INCLUDE="" # TYPE: SWITCH HELP: Include  ↩
     length field in length value [OFF] #
 LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.STRING="" # TYPE: OBJECT HELP: Use string format  ↩
     for the length field #
 LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.STRING.CHRSET="" # TYPE: NUMBER HELP: Character set ↩
      [SYSTEM] #
 LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.STRING.COUNT="" # TYPE: NUMBER HELP: Character  ↩
     count [C06] #
 LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.STRING.BASE="" # TYPE: NUMBER HELP: String base [ ↩
     B10] #
 LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.STRING.INCLUDE="" # TYPE: SWITCH HELP: Include  ↩
     length field in length value [OFF] #
 LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.BCD="" # TYPE: OBJECT HELP: Use BCD format for the  ↩
     length field #
 LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.BCD.WIDTH="" # TYPE: NUMBER HELP: Bit width/rate [ ↩
     B32] #
```

```
LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.BCD.INCLUDE="" # TYPE: SWITCH HELP: Include length field in length value [OFF] #
LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.HOST="" # TYPE: OBJECT HELP: Use host format (LLxx) for the length field #
LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.HOST.ENDIAN="" # TYPE: NUMBER HELP: Byte order [BIG ENDIAN] for first 16 Bit #
LIM.flcl.DIFF.COMPARE.RECORD.LENFORMAT.HOST.EXCLUDE="" # TYPE: SWITCH HELP: Record length field not part of the length value [OFF] #
LIM.flcl.DIFF.COMPARE.RECORD.KEYPOSITION="" # TYPE: NUMBER HELP: Key position (>=1) for index/key sequential data sets [0=NONE] #
LIM.flcl.DIFF.COMPARE.RECORD.KEYLENGTH="" # TYPE: NUMBER HELP: Key length (>=1) for index/key sequential data sets [if KEYPOS>0 then 8] #
LIM.flcl.DIFF.COMPARE.RECORD.DECODE="" # TYPE: NUMBER HELP: Decode encoded, encrypted or compressed data [0=NONE] #
LIM.flcl.DIFF.COMPARE.RECORD.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
LIM.flcl.DIFF.COMPARE.RECORD.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password decryption #
LIM.flcl.DIFF.COMPARE.RECORD.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to encrypt the FLAMFILE ['********'] #
LIM.flcl.DIFF.COMPARE.RECORD.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management extension (FKME) #
LIM.flcl.DIFF.COMPARE.RECORD.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for FKME ['libfkme'] #
LIM.flcl.DIFF.COMPARE.RECORD.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of FKME ['FKMESTD0'] #
LIM.flcl.DIFF.COMPARE.RECORD.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for the call of FKME [''] #
LIM.flcl.DIFF.COMPARE.RECORD.FDECODE="" # TYPE: OVERLAY HELP: Additional enforced decoding [NONE] #
LIM.flcl.DIFF.COMPARE.RECORD.FDECODE.BASE64="" # TYPE: SWITCH HELP: Base64 decoding ( RFC4648) #
LIM.flcl.DIFF.COMPARE.RECORD.FDECODE.BASE32="" # TYPE: SWITCH HELP: Base32 decoding ( RFC4648) #
LIM.flcl.DIFF.COMPARE.RECORD.FDECODE.BASE16="" # TYPE: SWITCH HELP: Base16 decoding ( RFC4648) #
LIM.flcl.DIFF.COMPARE.RECORD.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID ( enables text handling if set) #
LIM.flcl.DIFF.COMPARE.RECORD.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get wrong BOM in the middle of data #
LIM.flcl.DIFF.COMPARE.RECORD.SUBSYSTEM="" # TYPE: OBJECT HELP: Specification for FIO subsystem (only for z/OS) #
LIM.flcl.DIFF.COMPARE.RECORD.SUBSYSTEM.NAME="'FLAM'" # TYPE: STRING HELP: Name of the subsystem #
LIM.flcl.DIFF.COMPARE.RECORD.SUBSYSTEM.PARM="''" # TYPE: STRING HELP: Parameter list for the subsystem (comma separated) #
LIM.flcl.DIFF.COMPARE.RECORD.REMOVE="" # TYPE: SWITCH HELP: Remove original file after successful processing [FALSE] #
LIM.flcl.DIFF.COMPARE.RECORD.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG for read operation #
LIM.flcl.DIFF.COMPARE.RECORD.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable FL_PLATFORM for read operation #
LIM.flcl.DIFF.COMPARE.RECORD.HASH="" # TYPE: OBJECT HELP: Generation or check of one way hash values of input data #
LIM.flcl.DIFF.COMPARE.RECORD.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
LIM.flcl.DIFF.COMPARE.RECORD.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum verification #
LIM.flcl.DIFF.COMPARE.RECORD.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex- string #
LIM.flcl.DIFF.COMPARE.RECORD.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains Checksum as hex-string #
LIM.flcl.DIFF.COMPARE.RECORD.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [
```

```
    GNU] #
 LIM.flcl.DIFF.COMPARE.RECORD.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [ ↩
    LOG] #
 LIM.flcl.DIFF.COMPARE.RECORD.CHECK="" # TYPE: SWITCH HELP: Activate additional optional  ↩
    verifications [FALSE] #
 LIM.flcl.DIFF.COMPARE.FLAM4="" # TYPE: OBJECT HELP: Read records from a FLAM4FILE #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.FLAM4.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.COMPARE.FLAM4.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read ↩
     access #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.FLAM4.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.FLAM4.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.DIFF.COMPARE.FLAM4.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.COMPARE.FLAM4.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.FLAM4.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.FLAM4.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
    optional] #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.FLAM4.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.
```

```
The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.FLAM4.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.DIFF.COMPARE.FLAM4.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.COMPARE.FLAM4.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.FLAM4.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.FLAM4.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.FLAM4.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.FLAM4.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.COMPARE.FLAM4.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name  ↩
     [""] #
 LIM.flcl.DIFF.COMPARE.FLAM4.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.DIFF.COMPARE.FLAM4.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.FLAM4.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.DIFF.COMPARE.FLAM4.FILE="" # TYPE: STRING HELP: Name/URL of the FLAMFILE to read  ↩
     [required] #
 LIM.flcl.DIFF.COMPARE.FLAM4.MEMBER="" # TYPE: STRING HELP: Name or Index of the member in  ↩
     the FLAMFILE to read #
 LIM.flcl.DIFF.COMPARE.FLAM4.PRNCONTROL="" # TYPE: NUMBER HELP: Print control character  ↩
     handling [DETACH] #
 LIM.flcl.DIFF.COMPARE.FLAM4.RECCOUNT="" # TYPE: NUMBER HELP: Amount of records handled in  ↩
     one read operation [128] #
 LIM.flcl.DIFF.COMPARE.FLAM4.RECLENGTH="" # TYPE: NUMBER HELP: Maximal record length for  ↩
     read operation [512] #
 LIM.flcl.DIFF.COMPARE.FLAM4.TRUNCATE="" # TYPE: SWITCH HELP: Allow truncation of records [ ↩
     FALSE] #
 LIM.flcl.DIFF.COMPARE.FLAM4.PASSWORD="" # TYPE: STRING HELP: Passphrase to decrypt the  ↩
     FLAMFILE [''] #
 LIM.flcl.DIFF.COMPARE.FLAM4.KMLIBRARY="" # TYPE: STRING HELP: Library name for FKME [''] #
 LIM.flcl.DIFF.COMPARE.FLAM4.KMFUNCTION="" # TYPE: STRING HELP: Function name of FKME ['']  ↩
     #
 LIM.flcl.DIFF.COMPARE.FLAM4.KMPARAMETER="" # TYPE: STRING HELP: Parameter for the call of  ↩
     FKME [''] #
 LIM.flcl.DIFF.COMPARE.FLAM4.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID ( ↩
     enables text handling if set) #
 LIM.flcl.DIFF.COMPARE.FLAM4.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get  ↩
     wrong BOM in the middle of data #
 LIM.flcl.DIFF.COMPARE.FLAM4.REMOVE="" # TYPE: SWITCH HELP: Remove FLAMFILE after  ↩
```

```
    successful processing [FALSE] #
 LIM.flcl.DIFF.COMPARE.FLAM4.LANG="" # TYPE: STRING HELP: Redefine environment variable  ↩
    LANG for read operation #
 LIM.flcl.DIFF.COMPARE.FLAM4.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable ↩
     FL_PLATFORM for read operation #
 LIM.flcl.DIFF.COMPARE.FLAM4.HASH="" # TYPE: OBJECT HELP: Generation or check of one way  ↩
    hash values of input data #
 LIM.flcl.DIFF.COMPARE.FLAM4.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
 LIM.flcl.DIFF.COMPARE.FLAM4.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
 LIM.flcl.DIFF.COMPARE.FLAM4.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex- ↩
    string #
 LIM.flcl.DIFF.COMPARE.FLAM4.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
 LIM.flcl.DIFF.COMPARE.FLAM4.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [ ↩
    GNU] #
 LIM.flcl.DIFF.COMPARE.FLAM4.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [ ↩
    LOG] #
 LIM.flcl.DIFF.COMPARE.AUTO="" # TYPE: OBJECT HELP: Read binary, text or XML data record-  ↩
    or block-oriented #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.AUTO.NET:
With the overlay NET you can choose the protocol for the remote read
or write access. Below You can find the currently supported protocols
and the corresponding parameter explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.DIFF.COMPARE.AUTO.NET="" # TYPE: OVERLAY HELP: Network parameter for remote read  ↩
    access #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.AUTO.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.AUTO.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.DIFF.COMPARE.AUTO.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.COMPARE.AUTO.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.AUTO.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.AUTO.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
    optional] #
```

```
# DESCRIPTION for LIM.flcl.DIFF.COMPARE.AUTO.NET.IPS:
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.AUTO.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.DIFF.COMPARE.AUTO.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.DIFF.COMPARE.AUTO.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.DIFF.COMPARE.AUTO.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.AUTO.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

# DESCRIPTION for LIM.flcl.DIFF.COMPARE.AUTO.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.DIFF.COMPARE.AUTO.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.DIFF.COMPARE.AUTO.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name  ↩
     [""] #
 LIM.flcl.DIFF.COMPARE.AUTO.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.DIFF.COMPARE.AUTO.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.DIFF.COMPARE.AUTO.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [ ↩
     optional] #

 LIM.flcl.DIFF.COMPARE.AUTO.FILE="" # TYPE: STRING HELP: Name/URL of file to read [''== ↩
     stdin] #
 LIM.flcl.DIFF.COMPARE.AUTO.BLKSIZE="" # TYPE: NUMBER HELP: Block size to read [AUTO] #
 LIM.flcl.DIFF.COMPARE.AUTO.RECLENGTH="" # TYPE: NUMBER HELP: Maximum record length (Host)  ↩
     / line length (Unix/Win) for text parsing [512] #
 LIM.flcl.DIFF.COMPARE.AUTO.RPLFFD="" # TYPE: NUMBER HELP: Replace form feeds, filling rest ↩
      of page with blank lines assuming n lines per page [0 = no replacement] #
 LIM.flcl.DIFF.COMPARE.AUTO.CCSID="" # TYPE: STRING HELP: Conversion from this CCSID [auto  ↩
     detection] #
 LIM.flcl.DIFF.COMPARE.AUTO.BOM="" # TYPE: SWITCH HELP: Handle byte order change if get  ↩
```

```
    wrong BOM in the middle of data #
LIM.flcl.DIFF.COMPARE.AUTO.ENL2LF="" # TYPE: SWITCH HELP: Replace EBCDIC new line (0x15)  ↩
    by line feed (0x25) on input #
LIM.flcl.DIFF.COMPARE.AUTO.DECRYPT="" # TYPE: OVERLAY HELP: Decryption parameter [NONE] #
LIM.flcl.DIFF.COMPARE.AUTO.DECRYPT.F4PWD="" # TYPE: OBJECT HELP: Use FLAM password  ↩
    decryption #
LIM.flcl.DIFF.COMPARE.AUTO.DECRYPT.F4PWD.PASSWORD="" # TYPE: STRING HELP: Passphrase to  ↩
    encrypt the FLAMFILE ['********'] #
LIM.flcl.DIFF.COMPARE.AUTO.DECRYPT.F4KME="" # TYPE: OBJECT HELP: Use FLAM key management  ↩
    extension (FKME) #
LIM.flcl.DIFF.COMPARE.AUTO.DECRYPT.F4KME.LIBRARY="" # TYPE: STRING HELP: Library name for  ↩
    FKME ['libfkme'] #
LIM.flcl.DIFF.COMPARE.AUTO.DECRYPT.F4KME.FUNCTION="" # TYPE: STRING HELP: Function name of ↩
     FKME ['FKMESTD0'] #
LIM.flcl.DIFF.COMPARE.AUTO.DECRYPT.F4KME.PARAMETER="" # TYPE: STRING HELP: Parameter for  ↩
    the call of FKME [''] #
LIM.flcl.DIFF.COMPARE.AUTO.REMOVE="" # TYPE: SWITCH HELP: Remove original file after  ↩
    successful processing [FALSE] #
LIM.flcl.DIFF.COMPARE.AUTO.LANG="" # TYPE: STRING HELP: Redefine environment variable LANG ↩
     for read operation #
LIM.flcl.DIFF.COMPARE.AUTO.PLATFORM="" # TYPE: STRING HELP: Redefine environment variable  ↩
    FL_PLATFORM for read operation #
LIM.flcl.DIFF.COMPARE.AUTO.HASH="" # TYPE: OBJECT HELP: Generation or check of one way  ↩
    hash values of input data #
LIM.flcl.DIFF.COMPARE.AUTO.HASH.ALGO="" # TYPE: NUMBER HELP: Hash algorithm [SHA1] #
LIM.flcl.DIFF.COMPARE.AUTO.HASH.CHECK="" # TYPE: OVERLAY HELP: Parameter for checksum  ↩
    verification #
LIM.flcl.DIFF.COMPARE.AUTO.HASH.CHECK.VALUE="" # TYPE: STRING HELP: Checksum as hex-string ↩
     #
LIM.flcl.DIFF.COMPARE.AUTO.HASH.CHECK.FILE="" # TYPE: STRING HELP: File which contains  ↩
    Checksum as hex-string #
LIM.flcl.DIFF.COMPARE.AUTO.HASH.FORMAT="" # TYPE: NUMBER HELP: Define printout format [GNU ↩
    ] #
LIM.flcl.DIFF.COMPARE.AUTO.HASH.OUTPUT="" # TYPE: STRING HELP: Name of the output file [ ↩
    LOG] #
LIM.flcl.DIFF.COMPARE.AUTO.CHECK="" # TYPE: SWITCH HELP: Activate additional optional  ↩
    verifications [FALSE] #
LIM.flcl.DIFF.COMPARE.FILE="" # TYPE: STRING HELP: Abbreviation for read.auto(file=...) #
LIM.flcl.DIFF.LOGGING="" # TYPE: OVERLAY HELP: Select target for logging #
LIM.flcl.DIFF.LOGGING.STREAM="" # TYPE: OBJECT HELP: Log to a stream #
LIM.flcl.DIFF.LOGGING.STREAM.FORMAT="" # TYPE: NUMBER HELP: Format of log message [ ↩
    STANDARD] #
LIM.flcl.DIFF.LOGGING.STREAM.IDENT="" # TYPE: STRING HELP: Identifier used to identify log ↩
     entries #
LIM.flcl.DIFF.LOGGING.STREAM.STDOUT="" # TYPE: SWITCH HELP: Log messages to STDOUT instead ↩
     of STDERR #
LIM.flcl.DIFF.LOGGING.FILE="" # TYPE: OBJECT HELP: Log to a flat file #
LIM.flcl.DIFF.LOGGING.FILE.FORMAT="" # TYPE: NUMBER HELP: Format of log message [STANDARD] ↩
     #
LIM.flcl.DIFF.LOGGING.FILE.IDENT="" # TYPE: STRING HELP: Identifier used to identify log  ↩
    entries #
LIM.flcl.DIFF.LOGGING.FILE.NAME="" # TYPE: STRING HELP: Filename for appending log  ↩
    messages #
LIM.flcl.DIFF.LOGGING.SYSTEM="" # TYPE: OBJECT HELP: Use system logging facility #
LIM.flcl.DIFF.LOGGING.SYSTEM.IDENT="" # TYPE: STRING HELP: Identifier used to identify log ↩
     entries #
LIM.flcl.DIFF.MESSAGE="" # TYPE: OBJECT HELP: Select type of messages to log #
LIM.flcl.DIFF.MESSAGE.NONE="" # TYPE: SWITCH HELP: No messages will be logged #
LIM.flcl.DIFF.MESSAGE.ERROR="" # TYPE: SWITCH HELP: Error messages will be logged #
LIM.flcl.DIFF.MESSAGE.ERRTRACE="" # TYPE: SWITCH HELP: Error trace will be logged #
LIM.flcl.DIFF.MESSAGE.WARNING="" # TYPE: SWITCH HELP: Warnings will be logged #
LIM.flcl.DIFF.MESSAGE.NOTICE="" # TYPE: SWITCH HELP: Notices will be logged #
```

```
 LIM.flcl.DIFF.MESSAGE.DEBUG="" # TYPE: SWITCH HELP: Debug information will be logged #
 LIM.flcl.DIFF.MESSAGE.ABOUT="" # TYPE: SWITCH HELP: About information will be logged #
 LIM.flcl.DIFF.MESSAGE.VERSION="" # TYPE: SWITCH HELP: Version information will be logged #
 LIM.flcl.DIFF.MESSAGE.INPUT="" # TYPE: SWITCH HELP: Input data will be logged (dangerous  ↩
    for passwords) #
 LIM.flcl.DIFF.MESSAGE.PARAMETER="" # TYPE: SWITCH HELP: Parsed parameters will be logged #
 LIM.flcl.DIFF.MESSAGE.STATISTIC="" # TYPE: SWITCH HELP: Statistics will be logged #
 LIM.flcl.DIFF.MESSAGE.INFO="" # TYPE: SWITCH HELP: Requested information will be logged #
 LIM.flcl.DIFF.MESSAGE.LICID="" # TYPE: SWITCH HELP: License ID will be logged #
 LIM.flcl.DIFF.MESSAGE.LICENSE="" # TYPE: SWITCH HELP: License summary will be logged #
 LIM.flcl.DIFF.MESSAGE.PROGRESS="" # TYPE: SWITCH HELP: Progress bar will be logged #
 LIM.flcl.DIFF.MESSAGE.ALL="" # TYPE: SWITCH HELP: All messages are logged #
 LIM.flcl.DIFF.MESSAGE.DEFAULT="" # TYPE: SWITCH HELP: Relevant messages are logged #
 LIM.flcl.DIFF.MESSAGE.MINIMAL="" # TYPE: SWITCH HELP: Only important messages are logged #
 LIM.flcl.DIFF.MESSAGE.ERRONLY="" # TYPE: SWITCH HELP: Only errors and error trace are  ↩
    logged #
 LIM.flcl.DIFF.FULLY="" # TYPE: SWITCH HELP: Make a fully compare of data and elements [ ↩
    FALSE] #
 LIM.flcl.DIFF.OFFSET="" # TYPE: SWITCH HELP: Print byte offsets and values to report file  ↩
    [FALSE] #
 LIM.flcl.DIFF.REPORT="" # TYPE: STRING HELP: Generate a report file [NONE] #

# DESCRIPTION for LIM.flcl.UTIL.NET:
With the overlay NET you can choose the protocol for the remote read
access. Below, you can find the currently supported protocols and the
corresponding parameters explained in a dedicated chapter.

This feature requires a FLIES server.


 #
 LIM.flcl.UTIL.NET="" # TYPE: OVERLAY HELP: Network / Remote access #

# DESCRIPTION for LIM.flcl.UTIL.NET.IPN:
The IP native (IPN) method supports IPv4 and IPv6 network communication
for remote access to files, archives and other sources and targets of
the FLAM infrastructure. For this client-server architecture, the FLIES
server must be started and running in the IPN mode listening on a
dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish an IP native connection the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.UTIL.NET.IPN="" # TYPE: OBJECT HELP: IP-Native #

 LIM.flcl.UTIL.NET.IPN.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.UTIL.NET.IPN.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.UTIL.NET.IPN.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.UTIL.NET.IPN.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #

# DESCRIPTION for LIM.flcl.UTIL.NET.IPS:
```

```
The IP secure (IPS) method supports IPv4 and IPv6 network communication
with SSL/TLS for remote access to files, archives and other sources and
targets of the FLAM infrastructure. For this client-server architecture,
the FLIES server must be started and running in the IPS mode listening
on a dedicated port (17996) on the remote system to handle such
communication.

Optional user identification and authentication data can be provided.

To establish a secure IP connection, the host address (IP address or
DNS name) and the corresponding service port (number or name) must be
specified.

The default port for FLUC, FLAM and FLIES communication is 17996.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.UTIL.NET.IPS="" # TYPE: OBJECT HELP: IP-Secure (SSL/TLS) #

 LIM.flcl.UTIL.NET.IPS.HOST="" # TYPE: STRING HELP: IP address [localhost] #
 LIM.flcl.UTIL.NET.IPS.PORT="" # TYPE: STRING HELP: Port number [17996] #
 LIM.flcl.UTIL.NET.IPS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.UTIL.NET.IPS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #

# DESCRIPTION for LIM.flcl.UTIL.NET.MQS:
The MQ-Series (MQS) method supports IBM(R) message queuing for remote
access to files, archives and other sources and targets of the FLAM
infrastructure. For this client-server architecture, the FLIES server
must be started and running in MQS mode listening on a dedicated queue
on the remote system to handle such communication.

Optional user identification and authentication data can be provided.

For a remote access, the program acts as client. A corresponding FLIES
server is required to handle requests.


 #
 LIM.flcl.UTIL.NET.MQS="" # TYPE: OBJECT HELP: MQ-Series #

 LIM.flcl.UTIL.NET.MQS.QMANAGER="" # TYPE: STRING HELP: Queue manager name [""] #
 LIM.flcl.UTIL.NET.MQS.QUEUE="" # TYPE: STRING HELP: Queue name [""] #
 LIM.flcl.UTIL.NET.MQS.USER="" # TYPE: STRING HELP: User name [optional] #
 LIM.flcl.UTIL.NET.MQS.AUTH="" # TYPE: STRING HELP: Authentication data [optional] #

# DESCRIPTION for LIM.flcl.UTIL.RUN:
With the overlay RUN you choose which utility function to use. Be
careful with the remove operation for files. Please use the list
function first to see which files will get deleted when using wildcards.

All standard outputs (mainly the file list) of the utility functions
are written to STDOUT.

You can find the currently supported functions below. Don't hesitate to
request new functions that are useful with wildcard patterns.


 #
 LIM.flcl.UTIL.RUN="" # TYPE: OVERLAY HELP: Specify what you want to do #
```

```
 LIM.flcl.UTIL.RUN.LIST="" # TYPE: STRING HELP: List files #
 LIM.flcl.UTIL.RUN.REMOVE="" # TYPE: STRING HELP: Remove files #

# DESCRIPTION for LIM.flcl.UTIL.DIR:
This object defines multiple switches that control how input files are
found. This is especially important when using wildcards in input
filenames.

There are switches to enable recursion into subdirectories, following
symbolic links, finding hidden files, recursion into archive files,
include files on tapes, resolving aliases and enforcing to not leave
the filesystem.

If the recursive flag is set, the filename portion of the input file
string is searched for recursively. If a directory path containing
wildcards is present, the filename portion is searched for recursively
in all paths that match the directory path pattern, e.g. "*/path/*.txt"
matches my/path/hugo.txt, another/path/hugo.txt as well as
my/path/subdir/hugo.txt. Without the recursive flag, the last file
would not be found.

If the archive switch is enabled, FLAMFILES, TARBALLS, ZIP files and
other kinds of supported archives are searched. All the matching members
are extracted. In other words, archives are interpreted like
subdirectories if the flag is enabled.

If the tape switch is enabled, then datasets stored on tapes are
searched. If the name of such a file matches the pattern, then it might
occur that console input and a manual intervention is required to
retrieve the respective dataset(s).

If the link switch is enabled, symbolic links are followed (otherwise
they are ignored). Links are mainly supported on modern file systems.
For dataset aliases on mainframe systems, see the alias flag.

With the alias switch enabled, datasets names which are aliases of the
real dataset are resolved to their actual names (otherwise they are
ignored). Dataset aliases only exist on mainframe systems. This flag has
no effect on other platforms.

If a transparent read method (e.g. read.auto()) is used, then the
archive flag is enabled by default and cannot be disabled.


 #
 LIM.flcl.UTIL.DIR="" # TYPE: OBJECT HELP: Defines how to handle directories and various  ↩
     file types if wildcards used #

 LIM.flcl.UTIL.DIR.LINK="" # TYPE: SWITCH HELP: Include symbolic links #
 LIM.flcl.UTIL.DIR.ALIAS="" # TYPE: SWITCH HELP: Resolve data set aliases #
 LIM.flcl.UTIL.DIR.HIDDEN="" # TYPE: SWITCH HELP: Include hidden files #
 LIM.flcl.UTIL.DIR.RECURSIVE="" # TYPE: SWITCH HELP: Include sub directories #
 LIM.flcl.UTIL.DIR.ARCHIVE="" # TYPE: SWITCH HELP: Include archives (FLAMFILEs, ZIP, ...) #
 LIM.flcl.UTIL.DIR.ONEFILESYSTEM="" # TYPE: SWITCH HELP: Don't cross file system boundaries ↩
      #
 LIM.flcl.UTIL.DIR.TAPE="" # TYPE: SWITCH HELP: Include files deposited on tapes #

# DESCRIPTION for LIM.flcl.UTIL.LOG:
The LOG parameter allows flexible adjustment for the logging of
messages. It is possible to specify the destination as well as
which kind of message will be written.
```

```
For more information please have a look at the man pages for each
parameter of LOG.


 #
 LIM.flcl.UTIL.LOG="" # TYPE: OBJECT HELP: Logging Parameter #

# DESCRIPTION for LIM.flcl.UTIL.LOG.MESSAGE:
The log message specification is a classification of the messages to
write to the selected log destination. Each type is selectable
individually by adding its argument inside this object.

For added convenience, the special types NON, ALL, MINIMAL select
the proper message types with just one argument.

If no message specification is given, a DEFAULT is used which writes
out messages of these types:

 * ERROR
 * ERRTRACE
 * WARNING
 * NOTICE
 * PARAMETER
 * STATISTIC
 * INFO
 * LICID


 #
 LIM.flcl.UTIL.LOG.MESSAGE="" # TYPE: OBJECT HELP: Select type of messages to log #

 LIM.flcl.UTIL.LOG.MESSAGE.NONE="" # TYPE: SWITCH HELP: No messages will be logged #
 LIM.flcl.UTIL.LOG.MESSAGE.ERROR="" # TYPE: SWITCH HELP: Error messages will be logged #
 LIM.flcl.UTIL.LOG.MESSAGE.ERRTRACE="" # TYPE: SWITCH HELP: Error trace will be logged #
 LIM.flcl.UTIL.LOG.MESSAGE.WARNING="" # TYPE: SWITCH HELP: Warnings will be logged #
 LIM.flcl.UTIL.LOG.MESSAGE.NOTICE="" # TYPE: SWITCH HELP: Notices will be logged #
 LIM.flcl.UTIL.LOG.MESSAGE.DEBUG="" # TYPE: SWITCH HELP: Debug information will be logged #
 LIM.flcl.UTIL.LOG.MESSAGE.ABOUT="" # TYPE: SWITCH HELP: About information will be logged #
 LIM.flcl.UTIL.LOG.MESSAGE.VERSION="" # TYPE: SWITCH HELP: Version information will be  ↩
     logged #
 LIM.flcl.UTIL.LOG.MESSAGE.INPUT="" # TYPE: SWITCH HELP: Input data will be logged ( ↩
     dangerous for passwords) #
 LIM.flcl.UTIL.LOG.MESSAGE.PARAMETER="" # TYPE: SWITCH HELP: Parsed parameters will be  ↩
     logged #
 LIM.flcl.UTIL.LOG.MESSAGE.STATISTIC="" # TYPE: SWITCH HELP: Statistics will be logged #
 LIM.flcl.UTIL.LOG.MESSAGE.INFO="" # TYPE: SWITCH HELP: Requested information will be  ↩
     logged #
 LIM.flcl.UTIL.LOG.MESSAGE.LICID="" # TYPE: SWITCH HELP: License ID will be logged #
 LIM.flcl.UTIL.LOG.MESSAGE.LICENSE="" # TYPE: SWITCH HELP: License summary will be logged #
 LIM.flcl.UTIL.LOG.MESSAGE.PROGRESS="" # TYPE: SWITCH HELP: Progress bar will be logged #
 LIM.flcl.UTIL.LOG.MESSAGE.ALL="" # TYPE: SWITCH HELP: All messages are logged #
 LIM.flcl.UTIL.LOG.MESSAGE.DEFAULT="" # TYPE: SWITCH HELP: Relevant messages are logged #
 LIM.flcl.UTIL.LOG.MESSAGE.MINIMAL="" # TYPE: SWITCH HELP: Only important messages are  ↩
     logged #
 LIM.flcl.UTIL.LOG.MESSAGE.ERRONLY="" # TYPE: SWITCH HELP: Only errors and error trace are  ↩
     logged #

# DESCRIPTION for LIM.flcl.UTIL.LOG.DESTINATION:
The log destination parameter defines where the log messages will be
written.
```

```
Each log message starts with the current time and date.

All the possible destinations allow to specify a custom identification
string which will be printed after time and date of each message.
This is done with the IDENT argument.

If no IDENT specified the default will be 'owner.program.command'

If no log destination defined stream will be used as default.


 #
 LIM.flcl.UTIL.LOG.DESTINATION="" # TYPE: OVERLAY HELP: Destination for log messages #

# DESCRIPTION for LIM.flcl.UTIL.LOG.DESTINATION.STREAM:
The STREAM object allows the log messages to be written to the
standard output or the standard error output.
Standard error output is used if STDOUT is not specified.


 #
 LIM.flcl.UTIL.LOG.DESTINATION.STREAM="" # TYPE: OBJECT HELP: Log to a stream #

 LIM.flcl.UTIL.LOG.DESTINATION.STREAM.FORMAT="" # TYPE: NUMBER HELP: Format of log message  ↩
     [STANDARD] #
 LIM.flcl.UTIL.LOG.DESTINATION.STREAM.IDENT="" # TYPE: STRING HELP: Identifier used to  ↩
     identify log entries #
 LIM.flcl.UTIL.LOG.DESTINATION.STREAM.STDOUT="" # TYPE: SWITCH HELP: Log messages to STDOUT ↩
      instead of STDERR #

# DESCRIPTION for LIM.flcl.UTIL.LOG.DESTINATION.FILE:
The flat file option allows the log messages to be written to a normal
file. The name of the file must be specified for this destination.
The log messages will be appended to this file if it already exists.

For the log file, all rules and replacements are valid which are
described under "FILENAME HANDLING" above. For example:
```

```
NAME='~.MYLOG(LOG1)'                    ; PDS on z/OS
NAME='<SYSUID>.MYLOG02'                 ; PS dataset on z/OS
NAME='~/mylog3.txt'                     ; Unix path name
NAME='<HOME>\logs\mylog4.txt'           ; Windows path name
```

```
*ATTENTION:* Parallel process cannot share the same log file


 #
 LIM.flcl.UTIL.LOG.DESTINATION.FILE="" # TYPE: OBJECT HELP: Log to a flat file #

 LIM.flcl.UTIL.LOG.DESTINATION.FILE.FORMAT="" # TYPE: NUMBER HELP: Format of log message [ ↩
     STANDARD] #
 LIM.flcl.UTIL.LOG.DESTINATION.FILE.IDENT="" # TYPE: STRING HELP: Identifier used to  ↩
     identify log entries #
 LIM.flcl.UTIL.LOG.DESTINATION.FILE.NAME="" # TYPE: STRING HELP: Filename for appending log ↩
      messages #

# DESCRIPTION for LIM.flcl.UTIL.LOG.DESTINATION.SYSTEM:
The SYSTEM destination allows the log messages to be written to the
logging system of the operating system.

On Unix and z/OS(R) systems the syslog() call is used.
```

```
On Windows(R) the event logging facility is used.
```

Note
On Windows® this requires the registry key entry: HKEY_LOCAL_MACHINE SYSTEM CurrentControlSet Services EventLog Application FLM

```
This registry key will be created by using the DLL flevent.dll


 #
 LIM.flcl.UTIL.LOG.DESTINATION.SYSTEM="" # TYPE: OBJECT HELP: Use system logging facility #

 LIM.flcl.UTIL.LOG.DESTINATION.SYSTEM.IDENT="" # TYPE: STRING HELP: Identifier used to  ↩
     identify log entries #
```

Appendix D

RETURN CODES

The return codes of the program are also called completion codes and returned at program termination. This content is printed on the screen with the built-in function *ERRORS*.

- 0 - command line, command syntax, mapping, execution and finish of the command was successful
- 2 - command line, command syntax, mapping, execution was successful but cleanup of the command failed (may not happened)
- 4 - command line, command syntax and mapping was successful but execution of the command returned with a warning
- 8 - command line, command syntax and mapping was successful but execution of the command returned with an error
- 12 - command line and command syntax was OK but mapping of the parameter for command execution failed
- 16 - command line was OK but command syntax was wrong (error from command line parser)
- 20 - command string was wrong (please use built-in function "SYNTAX" for more information)
- 24 - initialization of the parameter structure for the command failed (may not happened)
- 28 - configuration is wrong (determination of configuration data failed, configuration file might be damaged)
- 32 - table error (something within the predefined tables is wrong (internal error))
- 36 - system error (e.g. memory allocation failed)
- 40 - access control or license error
- 44 - interface error (e.g. parameter pointer equals NULL (only at call of subprogram interface))
- 64 - fatal error (basic things are damaged)
- >64 - special condition codes from the command for job control

The command line is interpreted by CLE the command string by CLP. For example:

```
:> flcl conv "read.file='test.txt' write.flam(file='test.adc')"
   |    |     |-- the command string is compiled by CLP
   |    |     |   and might result in a syntax error
   |    |-- conv is part of the command line
   |-- flcl is part of the command line

:> flcl conv=para.txt
   |    |    |-- para.txt is part of the command line, but the content
   |    |       of para.txt might contain a CLP syntax error
   |    |-- conv is part of the command line
   |-- flcl is part of the command line
```

A command execution (run) will return 0 on success, 4 if a warning occurs, 8 in case of an error or special condition codes greater then 64 can be returned. If the command execution was successful (return code 0) a return code of 2 indicates that a call to the finish function failed. Such a fail of the finish function is not recognizable if the run function (command execution) returns with a warning or an error. All completion codes bigger than 8 and smaller or equal to 64 indicate an error in front of the command execution.

D.1 Special condition codes

Special condition codes are issued for a dedicated job/batch control. For error conditions, which must be handled in a special way, the condition codes below are defined.

- 112 - On charset detection, a text encoded in UTF-8 may erroneously be detected as an ASCII-based charset if the first block of data contains only ASCII characters (<128). In consequence, character conversion will eventually run into an "illegal sequence" (EILSEQ) error on the first multi-byte character which results in this condition code. In this case, it makes sense to re-run the command with the correct ASCII CCSID, for example Latin-1.

Appendix E

REASON CODES

Below you can find the list of reason codes returned after command execution. Each command returns a reason code for the corresponding warning (return code 4) or error (return code 8). This content is printed on the screen with the built-in function *ERRORS*.

- 1 - Fatal error (may not happen)
- 2 - System error (wrong size of data types)
- 3 - Handle error (not in right shape)
- 4 - Interface error (parameter incorrect)
- 5 - Activity not possible
- 6 - Memory allocation failed (not enough memory)
- 7 - Length error (encountered unexpected length)
- 8 - Control code wrong
- 9 - Version not supported
- 10 - Function not supported
- 11 - Mode not supported
- 12 - Suite not supported
- 13 - Confidentiality mode not supported
- 14 - Integrity mode not supported
- 15 - Verification mode not supported
- 16 - Wrong header checksum
- 17 - Wrong pot checksum
- 18 - Wrong data checksum
- 19 - Wrong checksum
- 20 - Wrong message authentication code (MAC)
- 21 - Wrong key test value
- 22 - Syntax error (file not formatted correctly)

- 23 - Count error (supplied or calculated amount wrong)
- 24 - End of list (handle points behind last or first element)
- 25 - Delete error (element already deleted)
- 26 - Type error (supplied or calculated type wrong)
- 27 - Rebuild required (flush not possible)
- 28 - No change (informational)
- 29 - Status error (handle not in right state)
- 30 - Orientation not supported
- 31 - Split method not supported
- 32 - Maximum delimiter amount achieved
- 33 - Error at compression
- 34 - Error at decompression
- 35 - Maximum header length achieved
- 36 - Maximum data length achieved
- 37 - Open of FLAMFILE failed
- 38 - Record format not supported
- 39 - Writing to file failed
- 40 - Reading from file failed
- 41 - Get position of file pointer failed
- 42 - Set position of file pointer failed
- 43 - End of file (informational)
- 44 - Rename of FLAMFILE failed
- 45 - Delete of FLAMFILE failed
- 46 - Close of FLAMFILE failed
- 47 - Seek in file does not work
- 48 - Set of buffer size for FIO failed
- 49 - FLAMFILE format not supported
- 50 - No FLAMFILE
- 51 - No entry found
- 52 - Construct is empty (informational)
- 53 - Error of IP native function
- 54 - Unsupported FLAM-ID
- 55 - Remote procedure call failed
- 56 - Method not supported
- 57 - Open of original file failed

- 58 - Error from ZLIB (GZIP)
- 59 - Character set not supported
- 60 - Space too small
- 61 - Logging facility not available
- 62 - Error from character conversion module
- 63 - Open of hash output file failed
- 64 - Close of original file failed
- 65 - Error with FKME
- 66 - Error at character conversion
- 67 - Info not supported
- 68 - Band not supported
- 69 - Open of report file failed
- 70 - Size error (supplied or calculated size wrong)
- 71 - Invalid character encountered
- 72 - Incomplete character encountered
- 73 - Error with byte order mark (BOM)
- 74 - Format wrong or not supported
- 75 - Auto conversion not possible
- 76 - Error from BZIP2LIB
- 77 - Environment variable are not or cannot be defined
- 78 - Dynamic load of a function failed
- 79 - Access control or license violation
- 80 - Compression limit not reached
- 81 - Error from LXZ
- 82 - Error from EXPAT
- 83 - Whole first block is padding
- 84 - Command line parser failed
- 85 - Mapping failed
- 86 - Open of temporary file failed
- 87 - Wrong magic bytes
- 88 - Corrupted FLAMFILE
- 89 - Initialization error
- 90 - Parameter not formatted correctly
- 91 - Binary data detected
- 92 - From and to are equal

- 93 - Data is encrypted (key required)
- 94 - Key value wrong
- 95 - Reallocation required
- 96 - Invalid username
- 97 - Authentication failed
- 98 - Hash calculation failed
- 99 - Processing of one or more files failed
- 100 - Hash/checksum file is empty
- 101 - Read of hash/checksum file failed
- 102 - Name of hash/checksum file is not valid
- 103 - Hash/checksum file is not formatted correctly
- 104 - Open of code table failed
- 105 - Code table is not formatted correctly
- 106 - Attribute settings not supported
- 107 - Option wrong or not supported
- 108 - Invalid parameter combination
- 109 - Open of info file failed
- 110 - Protocol wrong or not supported
- 111 - Differences found
- 112 - Try again with an ASCII code page (LATIN1) for read
- 113 - Data incomplete

Appendix F

VERSION

Here you can see the version of the program and its components. This content is printed on the screen if the built-in function *VERSION* is used.

```
00 FLCL VERSION: 5.1.8.10327 BUILD: RELEASE Aug 21 2015 20:16:06
01 FLUC VERSION: 5.1.8.10327 BUILD: RELEASE Aug 21 2015 20:16:15
02 FLUC-SUBPROGRAM VERSION: 5.1.8.10327 BUILD: RELEASE Aug 21 2015 20:16:06
03 FL-ORGLST VERSION: 5.1.8.10327 BUILD: RELEASE Aug 21 2015 20:17:10
04 FL-ORGNET VERSION: 5.1.8.10327 BUILD: RELEASE Aug 21 2015 20:17:10
05 FL-ORGIPN VERSION: 5.1.8.10327 BUILD: RELEASE Aug 21 2015 20:17:10
04 FL-ORGSAV VERSION: 5.1.8.10327 BUILD: RELEASE Aug 21 2015 20:17:10
05 FL-ORGFIL VERSION: 5.1.8.10327 BUILD: RELEASE Aug 21 2015 20:17:09
06 FL-ORGFIO VERSION: 5.1.8.10327 BUILD: RELEASE Aug 21 2015 20:17:09
07 FL-FIODAT VERSION: 5.0.3.10327 BUILD: RELEASE Aug 21 2015 20:15:55
07 FL-FIOBLK VERSION: 5.0.20.10327 BUILD: RELEASE Aug 21 2015 20:15:55
08 FL-FIO VERSION: 1.0.3.10327 BUILD: RELEASE Aug 21 2015 20:15:54
09 FL-BIOOPN VERSION: 1.0.11.10327 BUILD: RELEASE Aug 21 2015 20:15:42
09 FL-RIOOPN VERSION: 1.0.9.10327 BUILD: RELEASE Aug 21 2015 20:17:12
09 FL-TIOOPN VERSION: 1.0.8.10327 BUILD: RELEASE Aug 21 2015 20:17:14
07 FL-FIOREC VERSION: 5.0.17.10327 BUILD: RELEASE Aug 21 2015 20:15:58
08 FL-FIO VERSION: 1.0.3.10327 BUILD: RELEASE Aug 21 2015 20:15:54
09 FL-BIOOPN VERSION: 1.0.11.10327 BUILD: RELEASE Aug 21 2015 20:15:42
09 FL-RIOOPN VERSION: 1.0.9.10327 BUILD: RELEASE Aug 21 2015 20:17:12
09 FL-TIOOPN VERSION: 1.0.8.10327 BUILD: RELEASE Aug 21 2015 20:17:14
07 FL-FIOTXT VERSION: 5.0.14.10327 BUILD: RELEASE Aug 21 2015 20:15:58
08 FL-FIO VERSION: 1.0.3.10327 BUILD: RELEASE Aug 21 2015 20:15:54
09 FL-BIOOPN VERSION: 1.0.11.10327 BUILD: RELEASE Aug 21 2015 20:15:42
09 FL-RIOOPN VERSION: 1.0.9.10327 BUILD: RELEASE Aug 21 2015 20:17:12
09 FL-TIOOPN VERSION: 1.0.8.10327 BUILD: RELEASE Aug 21 2015 20:17:14
07 FL-FIOFL4 VERSION: 5.2.13.10327 BUILD: RELEASE Aug 21 2015 20:15:55
07 FL-FIOZIP VERSION: 5.1.1.10327 BUILD: RELEASE Aug 21 2015 20:15:59
08 FL-ZIP    VERSION: 1.0.0.10327 BUILD: RELEASE Aug 21 2015 20:17:14
09 FL-FIO VERSION: 1.0.3.10327 BUILD: RELEASE Aug 21 2015 20:15:54
10 FL-BIOOPN VERSION: 1.0.11.10327 BUILD: RELEASE Aug 21 2015 20:15:42
10 FL-RIOOPN VERSION: 1.0.9.10327 BUILD: RELEASE Aug 21 2015 20:17:12
10 FL-TIOOPN VERSION: 1.0.8.10327 BUILD: RELEASE Aug 21 2015 20:17:14
06 FL-ORGCNV VERSION: 5.1.8.10327 BUILD: RELEASE Aug 21 2015 20:17:08
07 FL-CNVBLK VERSION: 5.1.0.10327 BUILD: RELEASE Aug 21 2015 20:15:48
08 FL-FMTBIN VERSION: 5.0.2.10327 BUILD: RELEASE Aug 21 2015 20:16:18
08 FL-FMTBLK VERSION: 5.1.1.10327 BUILD: RELEASE Aug 21 2015 20:16:19
08 FL-FMTTXT VERSION: 5.0.21.10327 BUILD: RELEASE Aug 21 2015 20:16:19
07 FL-CNVREC VERSION: 5.1.0.10327 BUILD: RELEASE Aug 21 2015 20:15:51
08 FL-FMTTXT VERSION: 5.0.21.10327 BUILD: RELEASE Aug 21 2015 20:16:19
08 FL-FMTREC VERSION: 5.1.1.10327 BUILD: RELEASE Aug 21 2015 20:16:19
07 FL-CNVGZP VERSION: 5.1.17.10327 BUILD: RELEASE Aug 21 2015 20:15:50
```

```
07 FL-CNVBZ2 VERSION: 5.0.12.10327 BUILD: RELEASE Aug 21 2015 20:15:49
07 FL-CNVLXZ VERSION: 5.0.12.10327 BUILD: RELEASE Aug 21 2015 20:15:51
07 FL-CNVCHR VERSION: 5.1.5.10327 BUILD: RELEASE Aug 21 2015 20:15:49
08 FL-CHR    VERSION: 1.1.5.10327 BUILD: RELEASE Aug 21 2015 20:15:46
09 FL-ICV    VERSION: 1.0.5.10327 BUILD: RELEASE Aug 21 2015 20:16:49
09 FLDECT    VERSION: 1.0.10.10327 BUILD: RELEASE Aug 21 2015 20:15:53
07 FL-CNVBAS VERSION: 5.1.9.10327 BUILD: RELEASE Aug 21 2015 20:15:48
08 FL-BAS    VERSION: 1.0.0.10327 BUILD: RELEASE Aug 21 2015 20:15:42
07 FL-CNVCPY VERSION: 5.0.2.10327 BUILD: RELEASE Aug 21 2015 20:15:50
07 FL-CNVHSH VERSION: 5.0.4.10327 BUILD: RELEASE Aug 21 2015 20:15:51
07 FL-CNVDEC VERSION: 5.1.0.10327 BUILD: RELEASE Aug 21 2015 20:15:50
08 FL-CNVGZP VERSION: 5.1.17.10327 BUILD: RELEASE Aug 21 2015 20:15:50
08 FL-CNVBZ2 VERSION: 5.0.12.10327 BUILD: RELEASE Aug 21 2015 20:15:49
08 FL-CNVLXZ VERSION: 5.0.12.10327 BUILD: RELEASE Aug 21 2015 20:15:51
08 FL-CNVBAS VERSION: 5.1.9.10327 BUILD: RELEASE Aug 21 2015 20:15:48
09 FL-BAS    VERSION: 1.0.0.10327 BUILD: RELEASE Aug 21 2015 20:15:42
06 FL-ORGFMT VERSION: 5.1.8.10327 BUILD: RELEASE Aug 21 2015 20:17:10
07 FL-FMTBIN VERSION: 5.0.2.10327 BUILD: RELEASE Aug 21 2015 20:16:18
07 FL-FMTBLK VERSION: 5.1.1.10327 BUILD: RELEASE Aug 21 2015 20:16:19
07 FL-FMTREC VERSION: 5.1.1.10327 BUILD: RELEASE Aug 21 2015 20:16:19
07 FL-FMTTXT VERSION: 5.0.21.10327 BUILD: RELEASE Aug 21 2015 20:16:19
07 FL-FMTXML VERSION: 5.0.8.10327 BUILD: RELEASE Aug 21 2015 20:16:20
02 FL-DIROPN VERSION: 1.0.0.10327 BUILD: RELEASE Aug 21 2015 20:17:32
03 FL-FIOFL4 VERSION: 5.2.13.10327 BUILD: RELEASE Aug 21 2015 20:15:55
01 FLAM-CLE VERSION: 1.1.49.10327 BUILD: RELEASE Aug 21 2015 20:16:01
02 FLAM-CLP VERSION: 1.1.54.10327 BUILD: RELEASE Aug 21 2015 20:16:02
```

Appendix G

ABOUT

This is the about information for the program (flcl). Its content is printed on the screen if the built-in function *ABOUT* is used.

```
00 Frankenstein Limes Command Line (FLCL(R))
   Version: 5.1.8.10327 Build: RELEASE Aug 21 2015 20:16:06
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
The FLCL (R) is a universal command line interface to use the subprograms below:
    Frankenstein Limes Universal Converter (FLUC(R))
    Frankenstein Limes Access Method (FLAM(R))
    Frankenstein Limes Integrated Extened Security (FLIES(R))
01 Frankenstein Limes Universal Converter (FLUC)
   Version: 5.1.8.10327 Build: RELEASE Aug 21 2015 20:16:15
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This subprogram uses the libraries below:
02 Frankenstein Limes Universal Converter Subprogram (FLCUPM)
   Version: 5.1.8.10327 Build: RELEASE Aug 21 2015 20:16:06
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This subprogram uses the libraries below:
03 Frankenstein Limes original data list management (FL-ORGLST)
   Version: 5.1.8.10327 Build: RELEASE Aug 21 2015 20:17:10
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the libraries below:
04 Frankenstein Limes original data transfer (FL-ORGNET)
   Version: 5.1.8.10327 Build: RELEASE Aug 21 2015 20:17:10
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the libraries below:
05 Frankenstein Limes original data transfer per IP native (FL-ORGIPN)
   Version: 5.1.8.10327 Build: RELEASE Aug 21 2015 20:17:10
   Copyright (C) limes datentechnik  (R) gmbh
   All rights reserved
04 Frankenstein Limes original data storage (FL-ORGSAV)
   Version: 5.1.8.10327 Build: RELEASE Aug 21 2015 20:17:10
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the libraries below:
05 Frankenstein Limes original file management (FL-ORGFIL)
   Version: 5.1.8.10327 Build: RELEASE Aug 21 2015 20:17:09
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the libraries below:
```

```
06 Frankenstein Limes file IO for original data (FL-ORGFIO)
   Version: 5.1.8.10327 Build: RELEASE Aug 21 2015 20:17:09
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the libraries below:
07 Frankenstein Limes dummy file IO for data lists (FL-FIODAT)
   Version: 5.0.3.10327 Build: RELEASE Aug 21 2015 20:15:55
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
07 Frankenstein Limes block-oriented file IO (FL-FIOBLK)
   Version: 5.0.20.10327 Build: RELEASE Aug 21 2015 20:15:55
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library also uses the internal libraries below:
08 Frankenstein Limes File IO (FL-FIO)
   Version: 1.0.3.10327 Build: RELEASE Aug 21 2015 20:15:54
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library also uses the internal libraries below:
09 Frankenstein Limes Block IO for open platforms (FL-BIOOPN)
   Version: 1.0.11.10327 Build: RELEASE Aug 21 2015 20:15:42
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
09 Frankenstein Limes Record IO for open platforms (FL-RIOOPN)
   Version: 1.0.9.10327 Build: RELEASE Aug 21 2015 20:17:12
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
09 Frankenstein Limes Text IO for open platforms (FL-TIOOPN)
   Version: 1.0.8.10327 Build: RELEASE Aug 21 2015 20:17:14
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
08 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred  ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
07 Frankenstein Limes record-oriented file IO (FL-FIOREC)
   Version: 5.0.17.10327 Build: RELEASE Aug 21 2015 20:15:58
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library also uses the internal libraries below:
08 Frankenstein Limes File IO (FL-FIO)
   Version: 1.0.3.10327 Build: RELEASE Aug 21 2015 20:15:54
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library also uses the internal libraries below:
09 Frankenstein Limes Block IO for open platforms (FL-BIOOPN)
   Version: 1.0.11.10327 Build: RELEASE Aug 21 2015 20:15:42
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
09 Frankenstein Limes Record IO for open platforms (FL-RIOOPN)
   Version: 1.0.9.10327 Build: RELEASE Aug 21 2015 20:17:12
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
09 Frankenstein Limes Text IO for open platforms (FL-TIOOPN)
```

```
   Version: 1.0.8.10327 Build: RELEASE Aug 21 2015 20:17:14
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
08 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred ↩
      Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
07 Frankenstein Limes text-oriented file IO (FL-FIOTXT)
   Version: 5.0.14.10327 Build: RELEASE Aug 21 2015 20:15:58
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library also uses the internal libraries below:
08 Frankenstein Limes File IO (FL-FIO)
   Version: 1.0.3.10327 Build: RELEASE Aug 21 2015 20:15:54
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library also uses the internal libraries below:
09 Frankenstein Limes Block IO for open platforms (FL-BIOOPN)
   Version: 1.0.11.10327 Build: RELEASE Aug 21 2015 20:15:42
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
09 Frankenstein Limes Record IO for open platforms (FL-RIOOPN)
   Version: 1.0.9.10327 Build: RELEASE Aug 21 2015 20:17:12
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
09 Frankenstein Limes Text IO for open platforms (FL-TIOOPN)
   Version: 1.0.8.10327 Build: RELEASE Aug 21 2015 20:17:14
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
08 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred ↩
      Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
07 Frankenstein Limes FLAM4 file IO (FL-FIOFL4)
   Version: 5.2.13.10327 Build: RELEASE Aug 21 2015 20:15:55
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the FLAM4 record interface
07 Frankenstein Limes ZIP archive I/O (FL-FIOZIP)
   Version: 5.1.1.10327 Build: RELEASE Aug 21 2015 20:15:59
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library also uses the internal libraries below:
08 Frankenstein Limes ZIP archive module (FL-ZIP)
   Version: 1.0.0.10327 Build: RELEASE Aug 21 2015 20:17:14
   Copyright (C) limes datentechnik (R) gmbh
```

```
   All rights reserved
This library also uses the internal libraries below:
09 Frankenstein Limes File IO (FL-FIO)
   Version: 1.0.3.10327 Build: RELEASE Aug 21 2015 20:15:54
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library also uses the internal libraries below:
10 Frankenstein Limes Block IO for open platforms (FL-BIOOPN)
   Version: 1.0.11.10327 Build: RELEASE Aug 21 2015 20:15:42
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
10 Frankenstein Limes Record IO for open platforms (FL-RIOOPN)
   Version: 1.0.9.10327 Build: RELEASE Aug 21 2015 20:17:12
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
10 Frankenstein Limes Text IO for open platforms (FL-TIOOPN)
   Version: 1.0.8.10327 Build: RELEASE Aug 21 2015 20:17:14
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
06 Frankenstein Limes Conversion of original data (FL-ORGCNV)
   Version: 5.1.8.10327 Build: RELEASE Aug 21 2015 20:17:08
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the libraries below:
07 Frankenstein Limes block conversion (FL-CNVBLK)
   Version: 5.1.0.10327 Build: RELEASE Aug 21 2015 20:15:48
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the libraries below:
08 Frankenstein Limes Binary formatting (FL-FMTBIN)
   Version: 5.0.2.10327 Build: RELEASE Aug 21 2015 20:16:18
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
08 Frankenstein Limes Block formatting (FL-FMTBLK)
   Version: 5.1.1.10327 Build: RELEASE Aug 21 2015 20:16:19
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
08 Frankenstein Limes Text formatting (FL-FMTTXT)
   Version: 5.0.21.10327 Build: RELEASE Aug 21 2015 20:16:19
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library also uses the internal libraries below:
09 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred  ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
07 Frankenstein Limes record conversion (FL-CNVREC)
   Version: 5.1.0.10327 Build: RELEASE Aug 21 2015 20:15:51
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the libraries below:
08 Frankenstein Limes Text formatting (FL-FMTTXT)
   Version: 5.0.21.10327 Build: RELEASE Aug 21 2015 20:16:19
   Copyright (C) limes datentechnik (R) gmbh
```

```
   All rights reserved
This library also uses the internal libraries below:
09 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
08 Frankenstein Limes Record formatting (FL-FMTREC)
   Version: 5.1.1.10327 Build: RELEASE Aug 21 2015 20:16:19
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
07 Frankenstein Limes compression with ZLIB (FL-CNVGZP)
   Version: 5.1.17.10327 Build: RELEASE Aug 21 2015 20:15:50
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in unmodified form:
   zlib Version 1.2.8 Copyright (C) 1995-2012 Jean-loup Gailly and Mark Adler
   Thanks to Jean-loup and Mark to provide this software library
   for license see: http://www.gzip.org/zlib/zlib_license.html
This library uses the internal libraries below:
08 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
07 Frankenstein Limes compression with libbzib2 (FL-CNVBZ2)
   Version: 5.0.12.10327 Build: RELEASE Aug 21 2015 20:15:49
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in unmodified form:
   libbzip2 Version 1.0.6, 6-Sept-2010 Copyright (C) 1996-2007 Julian Seward
   Thanks to Julian to provide this software library
   for license see: http://www.bzip.org/
This library uses the internal libraries below:
08 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
07 Frankenstein Limes compression with liblzma LXZ (FL-CNVLXZ)
   Version: 5.0.12.10327 Build: RELEASE Aug 21 2015 20:15:51
```

```
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library liblzma Version 50020012.
   Thanks to Igor Pavlov and Lasse Collin to provide this software library
   for license see: http://www.tukaani.org/xz/
This library uses the internal libraries below:
08 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
07 Frankenstein Limes character set conversion (FL-CNVCHR)
   Version: 5.1.5.10327 Build: RELEASE Aug 21 2015 20:15:49
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the internal libraries below:
08 Frankenstein Limes character set conversion (FL-CHR)
   Version: 1.1.5.10327 Build: RELEASE Aug 21 2015 20:15:46
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the internal libraries below:
09 Frankenstein Limes character conversion module (FL-ICV)
   Version: 1.0.5.10327 Build: RELEASE Aug 21 2015 20:16:49
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
09 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
07 Frankenstein Limes Base16/32/64/85 conversion (FL-CNVBAS)
   Version: 5.1.9.10327 Build: RELEASE Aug 21 2015 20:15:48
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the internal libraries below:
08 Frankenstein Limes Base16/32/64 converter with ASCII and EBCDIC support (FL-BAS)
   Version: 1.0.0.10327 Build: RELEASE Aug 21 2015 20:15:42
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
08 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
```

```
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
07 Frankenstein Limes copy conversion (FL-CNVCPY)
   Version: 5.0.2.10327 Build: RELEASE Aug 21 2015 20:15:50
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
07 Frankenstein Limes conversion for hash calculation(FL-CNVHSH)
   Version: 5.0.4.10327 Build: RELEASE Aug 21 2015 20:15:51
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
07 Frankenstein Limes general decoding (FL-CNVDEC)
   Version: 5.1.0.10327 Build: RELEASE Aug 21 2015 20:15:50
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the libraries below:
08 Frankenstein Limes compression with ZLIB (FL-CNVGZP)
   Version: 5.1.17.10327 Build: RELEASE Aug 21 2015 20:15:50
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in unmodified form:
   zlib Version 1.2.8 Copyright (C) 1995-2012 Jean-loup Gailly and Mark Adler
   Thanks to Jean-loup and Mark to provide this software library
   for license see: http://www.gzip.org/zlib/zlib_license.html
This library uses the internal libraries below:
09 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
08 Frankenstein Limes compression with libbzib2 (FL-CNVBZ2)
   Version: 5.0.12.10327 Build: RELEASE Aug 21 2015 20:15:49
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in unmodified form:
   libbzip2 Version 1.0.6, 6-Sept-2010 Copyright (C) 1996-2007 Julian Seward
   Thanks to Julian to provide this software library
   for license see: http://www.bzip.org/
This library uses the internal libraries below:
09 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
08 Frankenstein Limes compression with liblzma LXZ (FL-CNVLXZ)
   Version: 5.0.12.10327 Build: RELEASE Aug 21 2015 20:15:51
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
```

```
This library is a work that uses the library liblzma Version 50020012.
   Thanks to Igor Pavlov and Lasse Collin to provide this software library
   for license see: http://www.tukaani.org/xz/
This library uses the internal libraries below:
09 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
08 Frankenstein Limes Base16/32/64/85 conversion (FL-CNVBAS)
   Version: 5.1.9.10327 Build: RELEASE Aug 21 2015 20:15:48
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the internal libraries below:
09 Frankenstein Limes Base16/32/64 converter with ASCII and EBCDIC support (FL-BAS)
   Version: 1.0.0.10327 Build: RELEASE Aug 21 2015 20:15:42
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
09 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
06 Frankenstein Limes formatting of original data (FL-ORGFMT)
   Version: 5.1.8.10327 Build: RELEASE Aug 21 2015 20:17:10
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the libraries below:
07 Frankenstein Limes Binary formatting (FL-FMTBIN)
   Version: 5.0.2.10327 Build: RELEASE Aug 21 2015 20:16:18
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
07 Frankenstein Limes Block formatting (FL-FMTBLK)
   Version: 5.1.1.10327 Build: RELEASE Aug 21 2015 20:16:19
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
07 Frankenstein Limes Record formatting (FL-FMTREC)
   Version: 5.1.1.10327 Build: RELEASE Aug 21 2015 20:16:19
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
07 Frankenstein Limes Text formatting (FL-FMTTXT)
   Version: 5.0.21.10327 Build: RELEASE Aug 21 2015 20:16:19
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library also uses the internal libraries below:
08 Frankenstein Limes detection modul(FLDECT)
   Version: 1.0.10.10327 Build: RELEASE Aug 21 2015 20:15:53
   Copyright (C) limes datentechnik (R) gmbh
```

```
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred  ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
07 Frankenstein Limes XML formatting (FL-FMTXML)
   Version: 5.0.8.10327 Build: RELEASE Aug 21 2015 20:16:20
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library is a work that uses the library below in modified form:
   expat Version expat_2.1.0 Copyright (C) 1998-2012 James Clark, Clark Cooper and Fred  ↩
       Drake
   Thanks to James, Clark and Fred to provide this software library
   for license see: http://expat.sourceforge.net/
   We fixed a few issues concerning our usage
   The modified package is published at the address below:
      https://github.com/limes-datentechnik-gmbh/expat
02 Frankenstein Limes Directory walk for POSIX / Windows (FL-DIROPN)
   Version: 1.0.0.10327 Build: RELEASE Aug 21 2015 20:17:32
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the internal libraries below:
03 Frankenstein Limes FLAM4 file IO (FL-FIOFL4)
   Version: 5.2.13.10327 Build: RELEASE Aug 21 2015 20:15:55
   Copyright (C) limes datentechnik (R) gmbh
   All rights reserved
This library uses the FLAM4 record interface
On LINUX systems we use additionally the glibc as part of the operating system
   FLCL(R) is a work that uses the library glibc in unmodified form based on the LGPL  ↩
       license
   Copyright (C) 1991-2012 Free Software Foundation, Inc.
   More information to this license you can find under:
      http://www.fsf.org/
      http://www.gnu.org/licenses/lgpl-3.0.txt
This program also uses the library below:
01 Frankenstein Limes Command Line Execution (FLAM-CLE)
   Version: 1.1.49.10327 Build: RELEASE Aug 21 2015 20:16:01
   Copyright (C) limes datentechnik (R) gmbh
   This library is open source from the FLAM(R) project: http://www.flam.de
   for license see: https://github.com/limes-datentechnik-gmbh/flamclep
This library uses the internal library below:
02 Frankenstein Limes Command Line Parser (FLAM-CLP)
   Version: 1.1.54.10327 Build: RELEASE Aug 21 2015 20:16:02
   Copyright (C) limes datentechnik (R) gmbh
   This library is open source from the FLAM(R) project: http://www.flam.de
   for license see: https://github.com/limes-datentechnik-gmbh/flamclep
```

Chapter 7

Index

A
Appendix About, 761
Appendix Grammar, 616
Appendix Lexem, 614
Appendix Properties, 744
Appendix Reasoncodes, 750
Appendix Returncodes, 746
Appendix Version, 752
Argument APPEND, 397
Argument ARMOR, 104, 107, 110, 137, 140, 143, 171, 174, 177, 206, 209, 213, 240, 244, 247, 349
Argument AUTO, 82, 515, 578
Argument BAS, 306, 347, 437
Argument BASE16, 108, 141, 175, 210, 245
Argument BASE32, 105, 138, 172, 207, 241
Argument BASE64, 101, 134, 168, 204, 238
Argument BCD, 68, 226, 283, 365, 414, 501, 564
Argument BIN, 309, 319, 440
Argument BINARY, 27, 90, 460, 524
Argument BLK, 274, 287, 310, 319, 327, 351, 405, 418, 441
Argument BLKSIZ, 275, 281, 406, 412
Argument BLKSIZE, 28, 35, 45, 54, 66, 83, 461, 468, 478, 487, 499, 516, 525, 532, 542, 551, 562, 579
Argument BOM, 37, 46, 55, 72, 79, 85, 122, 156, 191, 228, 257, 391, 470, 479, 488, 505, 512, 518, 533, 543, 552, 568, 575, 581
Argument BZ2, 292, 332, 423
Argument BZIP2, 97, 130, 164, 199
Argument CASE, 121, 155, 228, 256, 388
Argument CASMOD, 295, 336, 426
Argument CCSID, 36, 46, 55, 72, 78, 84, 121, 155, 191, 227, 256, 469, 479, 488, 505, 511, 517, 533, 543, 552, 568, 574, 580
Argument CHARACTER, 34, 115, 467, 531
Argument CHECK, 33, 34, 42, 43, 52, 61, 75, 76, 81, 89, 114, 148, 182, 217, 251, 265, 308, 351, 439, 457, 466, 467, 475, 476, 485, 494, 508, 509, 514, 522, 530, 531, 539, 549, 557, 572, 577, 585, 586
Argument CHR, 294, 334, 425
Argument CHRMODE, 122, 157, 192, 229, 257
Argument CNV, 287, 327, 418
Argument COMPARE, 523
Argument COMPRESS, 95, 128, 162, 198, 234
Argument DECRYPT, 28, 37, 47, 56, 69, 85, 461, 470, 480, 489, 502, 518, 525, 534, 544, 553, 565, 581
Argument DESTINATION, 23, 379, 400, 449, 455, 592
Argument DIR, 21, 266, 377, 398, 447, 453, 590
Argument DUMP, 326
Argument ELF2NL, 123, 156, 192, 300, 340, 392, 431
Argument ENCODE, 100, 133, 167, 203, 237
Argument ENCRYPT, 98, 131, 165, 201, 235
Argument ENL2LF, 37, 47, 56, 85, 300, 340, 392, 431, 470, 480, 489, 518, 534, 543, 552, 581
Argument ENV, 316, 376, 447
Argument F4KME, 29, 39, 48, 57, 70, 86, 100, 133, 167, 202, 237, 462, 472, 481, 490, 503, 519, 526, 535, 545, 554, 566, 583
Argument F4PWD, 29, 38, 48, 57, 70, 86, 99, 132, 166, 202, 236, 462, 471, 481, 490, 503, 519, 526, 535, 544, 553, 566, 582
Argument FALLOC, 91, 116, 150, 183, 220, 352, 359, 367, 382
Argument FDECODE, 30, 71, 463, 504, 527, 567
Argument FILE, 24, 90, 267, 273, 318, 380, 401, 404, 450, 456, 523, 586, 587, 593
Argument FIO, 273, 351, 404
Argument FKME, 20
Argument FL4, 286, 371, 417
Argument FLAM4, 76, 98, 131, 165, 200, 235, 252, 509, 573
Argument FMT, 309, 318, 440
Argument FROM, 296, 337, 389, 427
Argument GET, 19
Argument GZIP, 96, 129, 163, 198
Argument GZP, 291, 330, 422
Argument HASH, 32, 41, 51, 60, 74, 80, 88, 113, 147, 181, 216, 250, 264, 465, 474, 484, 493, 507, 513, 521, 529, 538, 548, 556, 571, 576, 584
Argument HDLBOM, 299, 340, 430
Argument HOST, 68, 226, 284, 366, 415, 501, 565
Argument HSH, 307, 349, 438
Argument IN, 382
Argument INPUT, 271, 402
Argument INTEGER, 67, 225, 282, 364, 413, 500, 563
Argument INVERSE, 269

Argument IPN, 272, 317, 403
Argument IPS, 272, 317, 403
Argument LANG, 40, 50, 58, 73, 79, 87, 146, 180, 215, 249, 263, 473, 483, 491, 506, 512, 520, 537, 546, 555, 569, 575, 583
Argument LENFMT, 282, 364, 413
Argument LENFORMAT, 67, 224, 500, 563
Argument LOG, 21, 377, 398, 448, 453, 591
Argument LOGGING, 266, 586
Argument LXZ, 292, 333, 423
Argument MESSAGE, 22, 268, 378, 399, 448, 454, 588, 591
Argument METHOD, 188, 295, 323, 335, 387, 426
Argument MINIMIZED, 190, 325
Argument MODE, 297, 337, 390, 428
Argument MQS, 273, 318, 404
Argument NET, 271, 316, 402
Argument OPGP, 111, 144, 178, 214, 248
Argument ORGPRN, 376
Argument OUT, 20, 382
Argument OUTPUT, 316
Argument PLATFORM, 31, 40, 50, 59, 73, 80, 87, 113, 146, 180, 215, 249, 263, 465, 473, 483, 492, 506, 513, 520, 528, 537, 547, 555, 570, 576, 583
Argument PRETTYPRINT, 189, 324
Argument PRNCONTROL, 30, 39, 49, 66, 77, 112, 145, 179, 227, 464, 472, 482, 499, 510, 527, 536, 545, 562, 573
Argument PRNCTR, 275, 281, 287, 356, 363, 406, 412, 418
Argument READ, 26, 460
Argument REC, 277, 289, 311, 320, 329, 357, 408, 420, 442
Argument RECORD, 62, 218, 387, 495, 558
Argument REPFIL, 305, 345, 436
Argument REPORT, 397
Argument REPORTFILE, 128, 162, 197, 234, 262
Argument RUN, 590
Argument SAV, 273, 318, 404
Argument SKPBIN, 305, 345, 436
Argument SKPEQU, 305, 346, 436
Argument SPACE, 94, 119, 153, 187, 223, 254, 355, 362, 370, 386
Argument SPLIT, 255
Argument STANDARD, 188, 323
Argument STREAM, 23, 266, 379, 400, 449, 455, 587, 593
Argument STRING, 67, 225, 283, 365, 414, 500, 564
Argument SUBCHAR, 124, 158, 193, 230, 258
Argument SUBCHR, 300, 341, 393, 431
Argument SUBSYS, 276, 284, 286, 407, 415, 417
Argument SUBSYSTEM, 31, 40, 49, 58, 73, 94, 119, 153, 186, 223, 355, 362, 370, 385, 464, 473, 482, 491, 506, 528, 536, 546, 554, 569
Argument SYSTAB, 301, 341, 393, 432
Argument SYSTABLE, 124, 158, 194, 230, 259
Argument SYSTEM, 24, 267, 380, 401, 450, 456, 588, 594
Argument TEXT, 44, 149, 477, 541
Argument TO, 298, 339, 392, 429
Argument TXT, 285, 313, 320, 366, 416, 444
Argument USRTAB, 303, 344, 396, 434
Argument USRTABLE, 127, 161, 196, 233, 261
Argument WRITE, 90
Argument WRTBOM, 299, 339, 430
Argument XML, 53, 183, 314, 322, 445, 486, 550
Argument XZ, 97, 130, 164, 200
Argument ZIP, 293, 424
Available built-in functions, 595
Available commands, 18

B

Built-in function ABOUT, 605
Built-in function CHGPROP, 599
Built-in function CONFIG, 603
Built-in function DELENV, 602
Built-in function DELPROP, 600
Built-in function ERRORS, 605
Built-in function GENDOCU, 597
Built-in function GENPROP, 598
Built-in function GETENV, 602
Built-in function GETOWNER, 601
Built-in function GETPROP, 600
Built-in function GRAMMAR, 603
Built-in function HELP, 596
Built-in function LEXEM, 604
Built-in function LICENSE, 604
Built-in function MANPAGE, 597
Built-in function SETENV, 601
Built-in function SETOWNER, 601
Built-in function SETPROP, 598
Built-in function SYNTAX, 595
Built-in function TRACE, 602
Built-in function VERSION, 604

C

COMMAND CONV, 26
COMMAND DIFF, 459
COMMAND HASH, 452
COMMAND ICNV, 381
COMMAND INFO, 19
Command line processor, 6
COMMAND UTIL, 590
COMMAND XCHK, 402
COMMAND XCNV, 271

D

Description for program flcl, 15

G

Glossary, 612

H

Help for program flcl, 17

S

Synopsis for program flcl, 7
Syntax for program flcl, 16

COLOPHON

limes datentechnik(R) gmbh
Louisenstrasse 21
D-61348 Bad Homburg v.d.H.
phone: +49(0)6172-5919-0
fax: +49(0)6172-5919-39
mail: info@flam.de
web: www.flam.de or www.limes.de
Amtsgericht: Bad Homburg vor der Hoehe HRB 3288 (gegr. 1985)
Geschaeftsfuehrer: Diplom-Mathematiker Heinz-Ulrich Wiebach
limes datentechnik(R): efficiency at the limit of possibility.